

Python

Pedro Luis Kantek Garcia Navarro
CEP

20 de abril de 2024

Sumário

1	Python	7
2	Instalação e alô mundo	9
2.1	Versão	9
2.2	sob Windows	9
2.3	Alo Mundo	10
2.4	Winpython	11
2.5	Ipython	11
2.6	Python na WEB	12
2.7	Jupyter	12
2.7.1	Vantagens	12
2.7.2	Desvantagens	12
3	Variáveis, sua entrada e saída	13
3.1	Atribuição	13
3.1.1	Atribuições malucas	14
3.2	Tipos	15
3.2.1	Mudando o tipo	16
3.2.2	Nomes	17
3.3	Expressões aritméticas	17
3.4	Saída de Dados	18
3.4.1	Opções da função print	18
3.5	Entrada de dados	19
3.5.1	Um programa completo usando input e print	19
3.6	Variáveis Lógicas	20
3.6.1	Expressões Lógicas	20
4	Comandos condicionais	23
4.1	Exercício	24
5	Repetições	29
5.1	while	29
5.2	break	29
5.2.1	Repetição aninhada	30
5.3	Continue	30
5.4	For	31
5.5	Exercícios	31
6	Listas	37
6.1	Fatiamento	38
6.1.1	fatiamento em arrays	39
6.2	Mais	40
6.3	Mais sobre listas	40
6.3.1	Usando listas como pilhas	41
6.3.2	Usando listas como filas	41
6.4	Ferramentas de programação funcional	42
6.5	List comprehensions ou abrangências de listas	43
6.5.1	Listcomps aninhadas	44
6.6	O comando del	44
6.7	Tuplas e sequências	45
6.8	Sets (conjuntos)	46
6.9	Dicionários	46

6.10	Técnicas de iteração	47
6.10.1	Métodos de dicionários	48
6.11	Mais sobre condições	49
6.12	Brincando com listas	49
7	Strings	53
7.0.1	String Methods	55
7.1	Formatação	56
7.1.1	Formatação de matrizes	57
7.1.2	String Formatting Operations	57
7.1.3	Manipulação de bits	58
8	Funções	59
8.1	Mais sobre definição de funções	60
8.1.1	Parâmetros com valores default	60
8.1.2	Listas arbitrárias de argumentos	62
8.1.3	Desempacotando listas de argumentos	62
8.1.4	Passagem por valor ou por referência	63
8.1.5	Construções lambda	63
8.1.6	Strings de documentação	63
9	Objetos em Python	65
9.1	Um exemplo de objeto: um formigueiro	66
9.1.1	Herança	67
10	Arquivos	69
10.1	Métodos de objetos arquivo	69
10.2	Um exemplo de criação de arquivo	71
10.3	O módulo pickle	71
11	Pacotes em Python	73
11.1	Conceito	73
11.2	Instalação	73
11.3	uma amostra	73
11.4	LISTA DE PACOTES PYTHON	73
12	Pacote: Numpy	77
12.1	Varejão	81
12.1.1	Para gravar arquivo de números	81
12.1.2	Para ler o arquivo acima	81
12.1.3	Para evitar a assunção de formato E na impressão	82
13	Pacote: Sympy	83
13.0.1	Exercício	86
13.1	Um exemplo de plotagem de uma função	86
13.1.1	Resposta	88
14	Pacote: Astropy	89
14.1	Padrão FITS	89
14.1.1	Formato	89
14.1.2	A cor no arquivo FITS	90
14.1.3	Como ver arquivos FITS	91
15	Pacote: Pygames e Turtle	93
16	Pacote: Matplotlib	97
16.1	Modos de uso	97
16.2	Para criar um gráfico simples, via métodos	97
16.3	O mesmo gráfico, via API	98
16.4	O método plot()	99
16.5	Um exemplo bobinho	100
16.6	Um exemplo de plotagem simples	100
16.6.1	Plotando pontos	101
16.7	Tortas	102

16.8	Histogramas	103
16.9	Um exemplo	104
16.10	Exemplo	105
16.11	Exemplo	106
16.12	Exemplo	107
16.13	Exemplo	108
16.14	Julia Set	109
16.15	Um exemplo de subplot	110
16.16	Interativo ou não	110
16.17	Escrevendo texto	111
16.18	Escrevendo latex	111
16.19	Mais texto	112
16.20	mais	112
16.21	Preenchimento	113
16.22	Exemplo com grid	114
16.23	Espiral de Fermat	114
16.24	Lemniscata	115
16.25	trifóide	116
16.26	cardióide	117
16.27	Anatomia de uma figura	119
17	Pacote: OpenCV	121
18	Pacote: TKInter	125
18.1	Hello World em Tkinter	125
18.2	Programa Inicial	125
18.3	Gerenciamento da geometria	126
18.4	Widgets	127
19	Pacote: TensorFlow	129
20	Pacote: Itertools	133
20.1	Funções do Pacote Itertool	134
21	UTF-8	137
21.1	Unicode	137
21.2	UTF=Unicode Transformation Formats	137
21.3	Exemplos	138
21.4	Exercício	139
21.4.1	Resposta	139
22	Um estudo de otimização	141
22.1	Otimização de algoritmos em Python	141
22.1.1	Primeira estratégia	141
22.1.2	Segunda estratégia	142
22.1.3	Terceira estratégia	142
22.1.4	Exercício	144
23	Pacote: Django	145
23.1	Passo a Passo: tutorial 1 da documentação	145
23.1.1	Obter e instalar os pacotes	145
23.1.2	Path	145
23.1.3	Criar o diretório do projeto Django	145
23.1.4	Começando	146
23.1.5	Testar com o browser	146
23.1.6	A qualquer momento	146
23.1.7	Criando sua primeira aplicação	146
23.1.8	Escrevendo a primeira coisa	147
23.1.9	Para testar	147
23.2	Passo a passo: tutorial 2	147
23.2.1	Estabelecendo o banco de dados	147
23.3	Passo a Passo: Tutorial 3	149
23.4	Passo a passo: Tutorial 4	149
23.5	Resumão	150

24 Tradução Automática	153
25 Leitura e geração de arquivos PDF	155
25.1 Capturando texto em PDF	158
26 Gerando arquivos PDF	159
27 LATEX: Inclusão de arquivos PDF	161
28 Análise de texto	163
29 WordCloud	165
30 Serviços Watson	167
30.1 Abrindo uma conta	167
31 Pillow: manuseando imagens	169
31.1 Instalação	169
31.2 Atributos da imagem	169
31.3 Separação e junção	170
31.4 Aplicando filtros	170
31.4.1 Cores	171
31.4.2 módulo ImageDraw	171
31.4.3 Imagem com numpy	171
32 Gerando aplicativo standalone	173
32.1 Receita de bolo em windows	173
32.1.1 Passagem de parâmetros	174
33 Bibliografia	175
33.1 Python	175
33.2 Algoritmos e programação	175
33.3 Métodos Numéricos	176

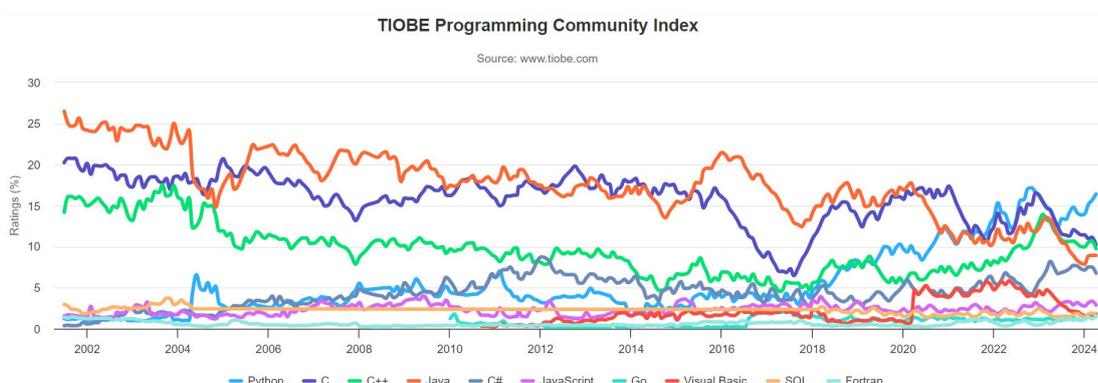
Capítulo 1

Python

A pergunta: **Porque Python ?**

- É uma linguagem moderna (nasceu em 1990) e herdou tudo de bom que as demais linguagens já tinham nessa época. Uma coisa que ela não herdou foi a preocupação exigente de desempenho: Em 90, já havia computação abundante. Com isso a linguagem é redonda e não impõe trancos e barrancos para funcionar. ¹
- Multiplataforma: Unix, Windows, Apple, celular android, raspberry e outros ambientes menos famosos rodam exatamente o mesmo programa.
- Freeware: pode copiar, baixar, instalar, vender, ceder... pode fazer qualquer coisa sem infringir a lei e sem dever explicações a ninguém.
- Uso no mainstream: Olhem o que diz o Philip Guo, um blogueiro da CACM "oito dos top-10 Departamentos de Ciência da Computação e 27 dos top-39 Departamentos de Ciência da Computação das principais universidades nos EUA usam Python nos cursos introdutórios de CS0 e CS1" (CS0 e CS1 são os cursos introdutórios de Computer Science lá). As 39 universidades pesquisadas por ele incluem: MIT, Berkeley, Stanford, Un. Columbia, UCLA, UIUC, Cornell, Caltech, Un. Michigan, Carnegie Mellon, entre outras ²
- Muito Usada: o índice TIOBE ³ lista o Python como a quinta linguagem mais usada no mundo em 2017. . Em junho de 2018, Python já é a quarta linguagem mais usada. Perde apenas para java, c e c++. E, corre por fora...
Se você olhar o estudo da TIOBE <https://www.tiobe.com/tiobe-index/> verá o bonito papel desempenhado pelo Python.

Situação em abril/24



¹Em compensação, quando se organiza uma maratona de programação, para um determinado problema, impõe-se 1 segundo de CPU quando ele é resolvido em C++ e pelo menos 10 (10 vezes mais) para quando ele é resolvido usando Python

²<https://cacm.acm.org/blogs/blog-cacm/176450-pyhton-is-now-the-most-popular-introductory-teaching-language-at-top-u-s-universities/fulltext>, acessado em agosto/2017

³www.tiobe.com, acessado em agosto/17

Linguagem	17	12	07	02	97	92
Java	1	1	1	1	12	-
C	2	2	2	2	1	1
C++	3	3	3	3	2	2
C#	4	4	7	17	-	-
Python	5	7	6	11	27	-
VB.NET	6	19	-	-	-	-
PHP	7	6	4	5	-	-
JavaScript	8	9	8	8	19	-
COBOL	25	28	17	9	3	10
Lisp	32	12	14	12	9	5
Prolog	33	32	26	15	20	12
Pascal	106	15	19	97	8	3

Consulta em 07/2017

Uma pequena explicação desta tabela: ela lista de 5 em 5 anos (o que no mundo da computação é quase uma eternidade) a lista dos top-10 ou as linguagens de programação mais usadas. O começo é 1992, e daí de 5 em 5 anos, por 30 anos ou 6 ciclos, até o último em 2017. Diversos fenômenos podem ser observados, eis alguns:

- A performance notável de Java, cuja explicação parece estar na palavra *portabilidade* ou seja o mesmo programa fonte rodando em inúmeras plataformas distintas. Isso é conseguido pelo conceito de java machine. Foge ao escopo deste texto descrever o conceito, mais em resumo é uma capa ou camada de compatibilidade que se adiciona ao ambiente de maneira a tornar ou permitir a mesma coisa sobre coisas diferentes.
- A derrocada do Pascal, nascido como uma excelente ferramenta de aprendizado e que em algum momento se perde completamente.
- A consistência de C e seus agregados (c++ e c#), seu sucesso pode ser descrito como eficiência e o possível alto nível de portabilidade.
- A morte anunciada e posteriormente executada do COBOL. Ele só permanece com alguma relevância graças ao legado ou aos milhões de linha de código que ainda precisam ser mantidos vivos.
- Quase a mesma coisa pode ser dita das linguagens da inteligência artificial: Lisp e Prolog. Elas migram para a generalidade: Java, C e Python.
- A irrupção da Internet: PHP, Javascript e Python, por trás dos panos. Inclusive da Internet das Coisas (aqui com Java).
- Pacotes, pacotes, pacotes. A comunidade Python mantém o PYPI (Python Package Index) que é um repositório freeware com pacotes de programação: hoje ele tem mais de 110.000 pacotes. Só para ter uma idéia, um pacote destes, o NUMPY (numerical computation in Python) tem um manual de referência de mais de 1500 páginas.
- Inteligência Artificial: Quando o Google resolveu aposentar o DistBelief e criar uma nova ferramenta de Machine Learning, escolheu a linguagem Python para desenvolvê-la: nasceu o Tensor Flow, que está por trás de inúmeras iniciativas google: tradução, classificação de imagens, elaboração de perfis etc. ⁴ Logo na entrada do produto o aviso: pode usar em Java, C++ ou Go. Mas, se quiser usar a funcionalidade completa do produto, use-o em Python, que ele está feito nessa linguagem.
- Fácil: parece português, ou vá lá: inglêsol. Não tem caracteres sobrando: sem ponto e vírgula, sem chaves nem colchetes. O que se escreve, ele executa e raramente dá erro.

⁴www.tensorflow.org

Capítulo 2

Instalação e alô mundo

Python é freeware, não custa nada, você pode copiar e usar para o que desejar, sem pedir licença a ninguém, e pode inclusive cobrar pelo software que você desenvolver usando Python. A partícula *free* em freeware significa liberdade para fazer o que você quiser sem nenhum tipo de restrição e não gratuidade, como alguns pensam.

Existem versões de Python para quase todos os sistemas operacionais conhecidos, eis alguns: AIX (da IBM), AROS (Amiga), AS/400, HPUnix, Linux, MacOS, MS-DOS, OS/2, PalmOS, Playstation, Solaris, VMS, Windows (32 e 64), WindowsCE, entre outros.

Esta multiplicidade de versões acrescenta uma qualidade imensa aos programas Python: desde que eles não usem nenhum recurso específico de uma dessas arquiteturas, serão 100% (ou quase isso) portáveis podendo rodar o mesmo programa em todas essas plataformas.

2.1 Versão

As versões de Python são numeradas em 3 níveis: O primeiro número identifica a versão global que atualmente pode ser 2 ou 3. Além das versões 0 e 1, quase experimentais, a linguagem começou sua carreira global com a versão 2, que chegou a ser bastante usada, muito software foi construído com ela, e esse fato impede agora que ela seja totalmente substituída pela versão 3, o que seria o melhor dos mundos. Se você está começando agora, não há o que pensar: deve-se escolher a versão 3. O problema é que na mudança de 2 para 3 perdeu-se alguma compatibilidade o que impede a livre transição de uma a outra. A versão 2 já está com sua sentença de morte estabelecida, ela só será suportada até 2020. Desse ponto em diante quem quiser usá-la o fará por sua conta e risco.

O segundo número indica a sub-versão, que diferencia algum nível maior de correção e sobretudo de lançamento de novidades. Finalmente o terceiro número vai sendo incrementado à medida em que pequenas correções e melhoramentos vão sendo construídos. Eis uma pequena versão das modificações:

Python 3.12	2 outubro 2023
Python 3.11	24 outubro de 2022
Python 3.10	4 outubro 2021
Python 3.9	5 outubro de 2020
Python 3.8	14 outubro 2019
Python 3.7	27 de junho de 2018
Python 3.6.5	28 Março 2018
Python 3.6.4	19 Dezembro 2017
Python 3.6.0	23 Dezembro 2016
Python 3.5.1	07 Dezembro 2015
Python 3.3.7	19 Setembro 2017
Python 3.2.6	11 Outubro 2014
Python 3.0	3 Dezembro 2008
Python 2.7.8	1 Julho 2014
Python 2.6.2	14 Abril 2009
Python 2.1.3	8 Abril 2002
Python 1.4	25 Outubro 1996

Existe tamanha diversidade no mundo Python que é impossível tratar todas as alternativas de uso. Primeiro, vai-se escolher o Python 3 e depois vai-se olhar 3 delas: a mais tradicional possível, sob Windows (nas 2 versões, sob 32 e sob 64 bits) e também uma instalação alternativa que permite instalar o Python em um pendrive para usá-lo em qualquer computador, mesmo sem os direitos de administrador nesse computador e o Python interativo. Ainda há a opção de usar o Jupyter.

2.2 Sob Windows

A primeira decisão é sob qual ambiente de windows você está. Uma maneira simples de descobrir é solicitar o *Painel de Controle*, e aqui solicitar a aba *System*. Isto mostra um texto com algumas características do sistema e lá no meio do texto, vem a frase “System type”. O que aparecer aí deve servir de guia para a escolha do Python a baixar e instalar: 32 ou 64 bits.

Daí, deve-se ir a <https://www.python.org/downloads/> e escolher a última versão de distribuição. Usualmente há versões mais novas que a última, mas são conhecidas como experimentais e não são recomendáveis, pois ainda não estão cristalizadas ou consolidadas. Deve-se escolher sempre o Python 3 e não o 2, que este vai desaparecer nos próximos anos e só está nessa lista para quem precisa usar e suportar software antigo. O Python 2.7 só receberá correções de segurança até 2025, mas desde hoje já não há novas funcionalidades.

No processo de instalação, especial preocupação deve ser tida ao verificar em qual diretório a instalação vai ser feita. Eu recomendo criar um diretório novo (não usar a sugestão do instalador) no primeiro nível, logo abaixo do nível raiz (como em `c:/python3`). Sob este diretório, o instalador vai criar diversos sub-diretórios (DLL, doc, etc, include, LIB, libs, Scripts, tcl e tools). É importante olhar isto, pois para instalar novos pacotes é preciso entrar no diretório Scripts, como se verá adiante. Logo é preciso saber onde está.

Depois de instalar o Python, deverá aparecer no seu menu inicial um ícone com 2 cobrinhas (uma azul e outra amarela)



o desenho padrão do Python. O nome deve identificar qual a versão instalada e quando se clicar nele deve-se abrir uma tela inicial do python, como em

```
Python 3.6.4 Shell
File Edit Shell Debug Options Window Help
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:
04:45) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for
more information.
>>> |
```

Nessa tela minimalista, surgem algumas informações como versão, data e hora da liberação desta versão, tipo de sistema sob qual ela está rodando. Depois algumas opções disponíveis e finalmente o símbolo

```
>>>
```

Ele é o *prompt* (prontidão) que é um conceito usado por praticamente todos os softwares interativos para sinalizar ao operador que o software está *pronto* para aceitar e executar comandos. Enquanto este símbolo não aparecer, isso significa que o processador está ocupado fazendo alguma coisa e não pode ser interrompido pelo operador.

Aqui já se podem escrever comandos Python como em

```
>>> 1+2
3
>>>
```

Aqui pediu-se ao Python a execução de $1 + 2$ e ele respondeu 3, seguido de um novo *prompt*. Esta modalidade de uso do Python é conhecida como interativa e neste caso ele funciona como se fosse uma calculadora. Basta comandar alguma coisa e ele a executará.

2.3 Alo Mundo

Tradicionalmente o primeiro programa que um novato deve construir em qualquer novo ambiente é chamado **Alo Mundo**. Sua função é apenas mostrar esta mensagem e sua execução com sucesso sinaliza que o processo de criar um programa fonte, salvá-lo, compilá-lo e depois executá-lo foi bem sucedido. Vamos a isso, então:

1. na tela do Python, escolha **File**
2. no menu que vai aparecer, escolha **New File** e lembre que estes dois passos podem ser comandados apertando **Ctrl N**
3. Vai-se abrir uma nova tela em branco, muito parecida com a tela do Python. Parece Python, mas não é Python e sim o bloco de notas do Python¹

¹Uma maneira enfiada de identificar qual tela é qual é esta: o bloco de notas do python tem o comando Run enquanto a tela do Python não tem este comando.

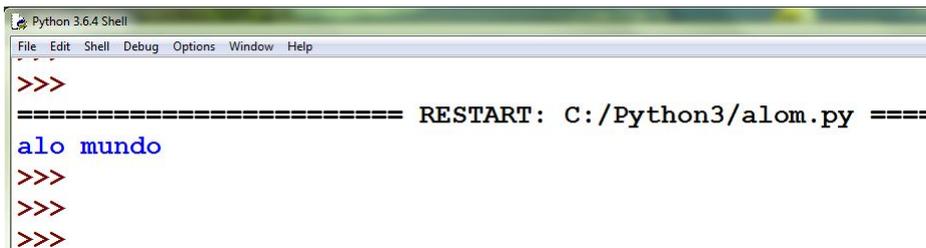
4. Na tela do bloco de notas do Python escreva o seguinte

```
print("alo mundo")
```

5. Agora você vai comandar a compilação (interpretação) e execução desse script (tecnicamente isto não é um programa e sim um script: é uma diferença semântica meio irrelevante, mas será estudada depois). Quem faz isto é a tecla **F5**. Ou também o comando **Run**, já destacado.

6. O Python antes de qualquer coisa vai pedir para você um nome sob o qual o programa (ou script) vai ser salvo. Esta é uma excelente exigência do Python: garante que aconteça o que acontecer, o programa que você escreveu e está tentando executar, será salvo em disco, sem risco de que ele se perca, caso o pior (tela azul da morte, por exemplo) aconteça. O nome que você der terá agregado o sufixo **.py** e será salvo no lugar onde você informar ²

7. Tendo sido salvo e compilado com sucesso a tela do Python vai mostrar



Esta tela mostra a execução do script que você criou e a mensagem sendo mostrada.

8. Isto significa que você teve sucesso na instalação do python e na criação e execução do seu primeiro programa.

2.4 Winpython

Esta implementação aparece aqui por algumas razões, sendo a principal a **portabilidade** do ambiente de desenvolvimento:

- Permite usar Python em máquinas nas quais você não é o administrador, coisa que o Python padrão exige para ser instalado.
- Como consequência, este Python é portátil, e pode ser carregado em um pendrive ou similar, ainda que haja alguma demora no processo de carga: o winpython é enorme.
- Winpython é um ambiente científico, pelo que já vem com SciPi e Numpy devidamente instalados, além de eventualmente outros pacotes importantes.

2.5 Ipython

É um python interativo, no qual fica mais evidente o diálogo entre usuário e Python. Na sua versão original (Python) é uma linguagem de programação que conta com um enfoque interativo também. No Ipython, ao contrário, é um ambiente interativo, que por acaso também dá acesso a uma linguagem de programação. Muitas vezes associado ao Jupyter, que é um ambiente interativo construído originalmente para as linguagens JULia, PYthon E (and) R. Posteriormente o ambiente foi migrado para muitas outras linguagens e hoje o menú é bem extenso. O ambiente é bom pois permite desenvolver notebooks que são arquivos simulando um caderno de notas mesclando texto, figuras e código.

Ele aparece aqui nesta lista pois é a versão disponibilizada nos laboratórios do curso de engenharia elétrica da UFPR. Neste ambiente, eis o caminho das pedras:

1. Usar seu login para entrar no sistema.
2. Na tela de atividades, digitar IP e aguardar. Escolher Ipython.
3. Carregar também o editor GEDIT. Manter ambos abertos. (Ipython e GEDIT)
4. Escrever o código python no GEDIT e salvar o programa e sua execução no diretório /eletrica, com o nome de aa.py
5. Dentro do Ipython executar **%run aa.py**
6. Para corrigir, edite no GEDIT, salve e reexecute dentro do Ipython.

²Sugestão para quem está começando: compre e use um pendrive para guardar seus programas: eles irão com você. E não esqueça de fazer cópias deles – o famoso backup.

2.6 Python na WEB

Uma alternativa disponível a quem tenha acesso à web e não pretenda (ou não possa) instalar nada em sua máquina é usar um ambiente python disponível na web. Há vários, mas um funcional é <https://www.pythonanywhere.com>. Aqui você precisa criar um login e depois disso passa a acessar um ambiente python completo. Já estão instalados nele vários pacotes importantes (teste: numpy, sympy, matplotlib, django). Este ambiente é perfeito para uma consulta rápida.

2.7 Jupyter

Este é um ambiente alternativo que permite a construção de texto e código python, tudo junto e misturado. Como tudo na vida tem vantagens e desvantagens. Eis algumas

2.7.1 Vantagens

- Permite escrever as duas coisas juntas: texto (como uma formatação muito simples, conhecida como Markdown. Veja a seguir os principais comandos) e código python. Este, pode ser facilmente compilado. Para escrever um livro de python não tem melhor.
- O material é facilmente compilado e impresso. Ele reside sob uma página HTML no seu computador.
- É instalar e sair usando, poucos minutos depois de começar.

2.7.2 Desvantagens

- A convivência entre Jupyter e o ambiente convencional de Python é complicada. Usar um programa que está dentro do Jupyter para uma programação de um sistema Python, praticamente implica em duplicar código: tudo de ruim aqui.

O texto em Jupyter é escrito dentro de células. Uma célula pode ser de texto (na cor verde) ou de código (na cor azul). Pode-se passar de uma a outra. Veja uma tabela (incompleta) de comandos:

Y	troca para código
M	troca para texto
CTRL-enter	executa a célula (lembre que ela é código...)
CTRL-A	seleciona tudo
A	insere uma célula acima
B	insere uma célula abaixo
X, C, V	cut, copy and paste
DD	deleta a célula
S	salva
#	título
...	itálico
*.	item em itemização
1.	item em enumeração

Para começar a usar o Jupyter, você precisa ter instalado o Python e nele deve fazer `pip install jupyter` como é usual. Este comando instala algumas dezenas de pacotes e depois disso, basta ir em uma janela DOS e executar `jupyter notebook` e sair usando.

Os arquivos do jupyter tem a terminação `ipynb` por mais estranho que possa parecer.

Capítulo 3

Variáveis, sua entrada e saída

Uma variável é um objeto do mundo do software que pode ter algumas abordagens. A primeira, é aquela da matemática, quando se escreve $y = f(x)$ e se quer dizer que y é uma variável que depende do valor de x . O que existe aí, nessa expressão, é uma função que associa o valor de y ao valor de x . Tanto x quanto y nesse enfoque são variáveis: isto é são valores eventualmente desconhecidos, mas que têm um valor qualquer. Outra abordagem é da ciência da computação: nela, uma variável é um pedacinho de memória dentro de um programa (que está devidamente registrado nessa memória) e que armazena um determinado valor. Cada variável precisa ter um nome, para que possa ser referenciada. Esse nome precisa ser único (não se pode ter duas variáveis com o mesmo nome), e depois que ela foi criada, usar essa variável num comando qualquer é exatamente a mesma coisa que usar o valor que a variável tem. Por exemplo, se a variável A tem o valor 10, tanto faz escrever $A + 1$ como $10 + 1$, só que no primeiro caso, a generalidade é maior, pois independentemente do valor que ela tiver, depois de executar $A + 1$, ter-se-á um valor uma unidade maior do que o valor anterior de A .

3.1 Atribuição

O comando importante aqui é o chamado **atribuição**. É ele que permite:

- criar uma variável
- alterar o valor de uma variável

As duas funções só se diferenciam pela existência prévia da variável. Se ela não existe, usar a atribuição significa criar a variável. Se ela já existia (e portanto tinha um valor qualquer) usar a atribuição nela vai alterar o valor dela para este, que é citado agora na atribuição.

O formato do comando é

variável = valor

Onde **variável** é o nome da variável, **=** identifica o comando de atribuição e **valor** é o valor que a variável passará a ter depois desta atribuição. Por exemplo:

A=10 A variável de nome A é criada (ou alterada) e passa a ter o valor 10.

X=-33.4 Agora o valor de X é o número negativo -33.4

NM='alfa' A variável agora se chama NM e seu valor é uma palavra, que é 'alfa'. Atente para a presença das aspas, elas se

Algumas observações importantes. O caráter de atribuição é **=** e ele é lido como **recebe**. Então, o comando $A = 10$ deve ser lido como **A recebe 10**. Outra observação é que este sinal tem outras funções no mundo. Por exemplo, na matemática o sinal de igual é isso mesmo: para indicar uma igualdade. Igualdade não é a mesma coisa que atribuição. O problema é que se não se usar o igual, que outro símbolo pode-se usar? Outras linguagens resolveram esse problema, usando dois caracteres (por exemplo, Pascal usa **:=**) enquanto outras usaram caracteres novos, como por exemplo APL, que usa **←**.

A discussão de qual símbolo usar para a atribuição (que lembrando é um conceito que não existe na matemática) é antiga e poderia ser aberta aqui. Mas, pegando carona no **C**, linguagem avô de boa parte do leque de linguagens usadas no século XXI, (tais como C++, Java, PHP, entre outras), o Python também associou o sinal de **=** à atribuição. A pergunta é: o que usar quando se falar sobre igualdade entre dois valores? Neste caso, usam-se 2 caracteres iguais. Então, para perguntar se as variáveis A e B têm o mesmo valor, escreve-se $A == B$, já que se se escrever $A = B$, o que se estará mandando fazer é atribuir o valor da variável B para a variável A .

Voltando ao formato do comando de atribuição, acima descrito, vamos estudar o que pode ser **valor** citado no formato do comando. Como o nome sugere, **valor** pode ser um número como 13, 2780.4, -33 e assim por diante. Mas, também pode ser outra variável, como em **A=X**. Aqui se está mandando copiar o valor de X para a variável A . Note que não se sabe que valor é esse. Não importa, ele é copiado de X para A .

Finalmente, esse valor pode ser uma expressão aritmética como em `A=11+8`, ou em `A=V-3`, ou finalmente `A=cos(theta)`. No primeiro caso, `A` vale 19, no segundo vale o valor de `V` menos 3 e no terceiro caso `A` vale o coseno do ângulo `theta`.

Agora pode-se ver a importância da presença das aspas no comando de atribuição: acompanhe no exemplo `A='alfa'` e `A=alfa`. No primeiro caso, é o conteúdo `alfa` que está sendo atribuído à variável `A`. No segundo pressupõe-se a existência de uma variável de nome `alfa` e é esta variável que tem seu valor (seja ele qual for) copiado para a variável `A`.

As aspas denotam um conteúdo alfanumérico, que estruturalmente é diferente de um conteúdo puramente numérico. Há muitas diferenças que ainda serão estudadas, mas uma importante é que conteúdos (e variáveis) numéricas podem ser usadas em expressões aritméticas enquanto variáveis alfanuméricas causam erros de execução quando se tenta envolvê-las em expressões numérico-aritméticas.

```
>>> A=5.5      Cria-se a variável A, flutuante, com o valor 5.5
>>> B='oba'    Cria-se B alfanumérica contendo 'oba'
>>> A+B        Tenta-se somar as duas variáveis. Há um erro
```

O erro é sinalizado pela mensagem `TypeError: unsupported operand type(s) for +: 'float' and 'str'` cuja tradução é: Erro de tipo: não se pode somar (+) um tipo float com um tipo str.

3.1.1 Atribuições malucas

O Python, na esteira das linguagens mais modernas, tem diversas alternativas de atribuição, que serão vistas agora. Na minha opinião, este capítulo pode ser ignorado, pois os acréscimos que serão vistos aqui, embora acrescentem riqueza ao ambiente, muitas vezes são únicos do Python e ao usá-los, afastamo-nos da facilidade de transpor nosso código para outras linguagens (C++ ou Java, as maiores candidatas). Eu considero transposição importante, cada um que julgue se deve ou não priorizá-la.

Múltiplo assinalamento

Pode-se atribuir o mesmo valor a diversas variáveis:

```
a=b=c=d=0
```

Neste caso, foram criadas (alteradas) as variáveis `a`, `b`, `c` e `d` todas com o valor zero.

Múltiplo assinalamento com diversos valores

O valor não precisa ser o mesmo, pode variar. Veja no exemplo

```
>>> a,b,c=5.6,0,'alfa'
>>> a
5.6
>>> b
0
>>> c
'alfa'
```

Note que a quantidade de variáveis à esquerda e de expressões à direita precisa ser a mesma, sob pena de um erro como em

```
a,b,c=1,2,3,4
ValueError: too many values to unpack (expected 3)
```

Swap de variáveis

Em programação muitas vezes é necessário inverter o conteúdo de 2 variáveis. Em situação normal, isso exige o uso de uma variável auxiliar como em

```
>>> aux=a
>>> a=b
>>> b=aux
```

Python possui um mecanismo simples para realizar a troca sem usar variáveis adicionais. O mesmo exemplo acima pode ser feito assim

```
a,b=b,a
```

Cabeça e cauda

Em muitos ambientes de programação, notadamente em programação funcional, baseada em listas, como na linguagem LISP (List Processor) e mesmo na linguagem PROLOG (Processador Lógico) ambas muito usadas no reino da Inteligência artificial, há uma estratégia de programação, fortemente baseada em recursividade e que exige processar um elemento da lista de cada vez, diminuindo-a sempre em direção à lista vazia, que neste caso é o que se chama de *caso básico* (para entender melhor esta abordagem, veja algum texto sobre recursividade). Vou descrever o funcionamento disto como em Lisp, que foi quem começou com essa abordagem. Dada uma lista, por exemplo `[10,20,30,40]`, ela sempre pode ser dividida em cabeça (que em lisp se chama CAR) e corpo (que em lisp se chama CDR). A cabeça é o primeiro elemento da lista e o corpo é o que sobra quando a cabeça é eliminada da lista. Em lisp

```
> (car '(10 20 30 40))
10
> (cdr '(10 20 30 40))
(20 30 40)
```

Em LISP a cabeça é sempre o primeiro, em Python pode ser o primeiro ou o último. Veja na implementação

```
>>> seq=[10, 20, 30, 40]
>>> a,*b=seq
>>> a
10
>>> b
[20, 30, 40]

>>> *a,b=seq
>>> a
[10, 20, 30]
>>> b
40
```

Assinalar e operar ao mesmo tempo

Em programação é muito frequente escrever `a=a+1`. Este comando em Python pode ser escrito como `a+=1`. Note que o perigo reside – numa distração – em inverter os dois sinais. Ao se escrever `a=+1` está-se comandando outra coisa: a criação (ou alteração) da variável `a` com o valor de +1. Veja o perigo: você cometeu um erro, e o Python não avisa, faz o que você mandou (como sempre), mas eventualmente não era isso que você queria.

O par de sinais pode ser `+=`, além de `--`, `*=`, `/=`, `%=`, `//=`, `**=` entre outros.

Exclusão de variável

Embora não seja usual, o programador pode excluir as variáveis quando estas deixarem de ser úteis. Não é usual, pois ao encerrar o Python automaticamente tudo o que foi usado é excluído e os recursos liberados. Mas, havendo necessidade o programador pode excluir variáveis com o comando `del`. Veja-se

```
>>> a=5
>>> b=6
>>> del a,b
>>> a
NameError: name 'a' is not defined
```

3.2 Tipos

Graças a essa distinção vista acima, pode-se pontuar a respeito de uma característica importante das variáveis. Cada variável terá um tipo que sinalizará como ela será manipulada depois de criada. Os principais tipos são: numérico inteiro, numérico real, alfanumérico e lógica.

Ao contrário de outras linguagens (como o C, por exemplo), as variáveis em Python não têm um tipo explicitamente declarado. É o seu valor que determina qual é o seu tipo. ¹ Então,

<code>A=10.3</code>	A é uma variável do tipo real, que em Python é conhecida como <code>float</code>
<code>B=5</code>	A variável <code>B</code> é do tipo inteiro.
<code>C=10.0</code>	A variável <code>C</code> é tipo float
<code>HAL='ivo viu a uva'</code>	A variável HAL contém a frase <code>ivo viu a uva</code>
<code>GA=True</code>	GA é uma variável do tipo lógico que tem o valor de verdadeiro
<code>J=False</code>	J é uma variável do tipo lógico e tem o valor Falso

¹Esta regra nem sempre é verdadeira, por exemplo, ao usar o pacote Numpy. Mas, por enquanto pode-se aceitá-la como verdadeira.

A variável do tipo lógico só admite dois valores `True` e `False`. Obviamente, estes dois nomes não servem para dar nomes a variáveis. É isto que se chama de palavra reservada .

Os tipos numéricos podem ser inteiros (`int`) ou reais, que no Python recebem o nome de `float`. Poder-se-ia argumentar que o tipo inteiro é desnecessário (já que o conjunto dos reais contém o conjunto dos inteiros ou $\mathbb{R} \subset \mathbb{Z}$), mas isto não é verdade, já que:

- Dentro do computador, a matemática inteira é dezena de vezes mais rápida que a matemática real. Então quando aplicável, sempre vale a pena ser mais eficiente.
- Mas, principalmente, o conjunto dos inteiros tem uma característica que o real não tem: a enumerabilidade. Tal característica é muito importante (por exemplo, na indexação) e ela exige que o tipo seja inteiro em certos momentos.

Feita esta distinção, os conteúdos que uma variável numérica pode ter são:

inteiro : os 10 dígitos `0,1,2,3,4,5,6,7,8 e 9`, um sinal opcional `+` ou `-`.

flutuante : os mesmos dígitos e sinal e além destes, o sinal de ponto decimal. Note que usa-se em Python o ponto decimal e não a vírgula decimal, herança dos países de língua inglesa. Opcionalmente pode-se ter a letra `E` ou `e` significando exponencial. `E`, se presente, indica o expoente de 10, que deve ser multiplicado pelo número antes do `E`. O expoente também pode ter opcionalmente um sinal que pode `+` ou `-`. Exemplos:

```
3E4   Significa  $3 \times 10^4$  ou 3000
-4E2   $-4 \times 10^2$  ou -400
5E-3  $5 \times 10^{-3}$  ou  $5 \times \frac{1}{10^3}$  ou ainda  $5 \times 0.001$  que é 0.005.
```

3.2.1 Mudando o tipo

O tipo pode ser modificado, por exemplo em

```
A=9           Cria-se a variável A como inteiro
B=6.5        Cria-se B como flutuante
A=B          A que era inteiro passa a ser flutuante, pois recebeu 6.5
A='Curitiba' A agora é uma variável string
A=False      Finalmente, A agora é uma variável lógica
```

Nenhuma dessas modificações causa algum tipo de erro ou de manifestação do Python. São ações normais e corriqueiras.

Em alguns momentos da programação, é necessário explicitamente modificar um tipo de variável. Esta necessidade já vai aparecer (ao tratar o comando `input`, por exemplo), mas ela vai ser vista aqui do ponto de vista formal. Para mudar o tipo de uma variável pode-se usar

```
int(x)       Pega o conteúdo de x e transforma-o (se possível) em um inteiro.
float(x)     Transforma x em flutuante
str(x)       Transforma x em uma string
```

Obviamente a transformação só vai acontecer se ela for possível. Não adianta, por exemplo, definir `A='alfa'` e pedir `int(A)`, já que 'alfa' não pode ser transformada em um número inteiro. Mas, se `B='123'` deve-se notar que `B` é uma variável string (por causa das aspas), mas ela pode ser transformada em inteira e neste caso, valerá 123, pois só é formada por dígitos numéricos. Já a conversão em string é sempre possível, e sempre que for ordenada será realizada.

Para a representação de números inteiros o Python usa um sistema de precisão ilimitada, que permite usar muitos (muitos !) dígitos, veja-se por exemplo o fatorial de 50 é

```
30414093201713378043612608166064768844377641568960512000000000000. O programador não precisa fazer nada (ao contrário de outras linguagens mais antigas como C ou C++). O Python simplesmente lança mão de quantas casas forem necessárias para a representação numérica inteira pedida.
```

Já na representação flutuante, podem surgir problemas derivados do fato de que o Python usa para representação interna um formato binário limitado composto de abcissa e mantissa. Em outras palavras, um número flutuante usa uma quantidade finita de bits para ser representado, o que faz com que haja perda de precisão nessa representação. Veja-se um exemplo prático (uma sessão real de Python)

```
>>> a=0.1
>>> print("{:2.5f}".format(a))
0.10000
>>> print("{:2.30f}".format(a))
0.1000000000000000005551115123126
```

O comando format acima, parece, e é, bem misterioso, mas o que ele pede é algo prosaico, que a variável a seja impressa, no primeiro caso, com 5 decimais e no segundo com 30 decimais. Observe o que aconteceu. O surgimento de números (aparentemente aleatórios) a partir da 18ª casa é resultado de imprecisões derivadas do uso do tipo flutuante. Perceba-se que não deve haver maiores problemas: um erro na 18ª casa não é para preocupar mas, ele está lá.

3.2.2 Nomes

A regra para dar nomes em Python (regras essas que se aplicam à nomeação de variáveis) são simples:

- uma única palavra, ou seja não se admitem espaços em branco dentro dos nomes.
- pode-se usar o caracter sublinha (`_`) para separar palavras no nome
- o primeiro caractere deve ser uma letra
- depois do primeiro pode-se usar letras ou números
- Letras maiúsculas são diferentes das letras minúsculas (ou seja `OBA` é diferente de `Oba` que não é a mesma coisa que `oba`).

O leitor não deve se preocupar muito com estas regras, substituindo-as por algum bom senso: os nomes devem ser autodeclarativos (sobre para que serve a variável), com nomes nem muito grandes (provocam erros de digitação) nem muito pequenos (para que serve mesmo a variável B ?). Uma regra muito usada por aí sugere usar várias palavras ou abreviaturas no nome, separando umas das outras pelo uso de maiúsculas. Usando esta regra pode-se ter

`SaldoConta` uma variável para conter o saldo da conta
`LucroAntesIR` auto-explicativo

Os detratores da regra acima dizem que não se deve misturar maiúsculas e minúsculas no nome, pois isto acaba sendo uma fonte de erros de digitação. Estes dizem que as variáveis devem ser todas minúsculas ou todas maiúsculas.

Enfim, a sugestão aqui é que cada leitor deve definir a sua regra de formação de nomes e se ater a ela, sempre visando a minimização dos erros de digitação.

A partir da versão 3, o Python pode aceitar caracteres acentuados na formação de nomes, pois aqui por padrão usa-se o conjunto de caracteres Unicode, através do Unicode Transformate Format-8 (UTF-8). Mas, pensando francamente, não parece ser uma boa idéia: se você definir uma variável de nome `SAÍDA` e depois chamá-la de `SAIDA`, para o Python serão duas variáveis distintas, e portanto aqui está uma nascente fonte de problemas. Tente não usar caracteres acentuados ao programar. Você não vai se arrepender. Conteúdos dentro de aspas (constantes) podem e devem usar caracteres acentuados: vamos homenagear a Língua Portuguesa, e dentro das aspas não se corre nenhum perigo. Como em `ESTADO='Paraná'`.

3.3 Expressões aritméticas

As variáveis e constantes numéricas podem ser operadas e interligadas usando-se as operações aritméticas usuais (e algumas não tão usuais) da matemática convencional. Para isso são usados os símbolos a seguir descritos, que buscam seguir – tanto quanto possível – as convenções usuais da escrita matemática. Por exemplo

```
>>> a = 5
>>> b = 3.6
>>> a + 4
9
>>> b * 3
10.8
>>> (a + b) / 2
4.3
>>> a - 6
-1
>>> a - -8
13
```

Note que a adição é representada pelo sinal de “mais” (+), a subtração por “menos” (–), a multiplicação pelo asterisco (*) e a divisão real pela barra (/).

Note também que a prioridade é determinada pelo uso de parênteses, que as vezes é redundante, mas nem sempre, pois se o comando fosse escrito como $2 + 8/2$ sem parênteses, o Python primeiro faria a divisão de $8/2$ dando 4 e depois a soma de 2 com 4 resultando 6. Ao se escrever $(2 + 8)/2$ primeiro a soma dando 10 e depois a divisão por 2, que dará 5 de resposta final. A seguir, uma tabela de algumas operações aritméticas:

Símb	Nome	Como funciona
+	adição	em $a + b$ o resultado é a adição entre a e b .
-	subtração	em $a - b$ o resultado é $a - b$.
*	multiplicação	Em $a * b$ o resultado é a multiplicação de a e b .
/	divisão real	Em a/b o resultado é um float contendo a divisão de a por b .
//	divisão inteira	Em $a//b$ o resultado é um inteiro contendo a divisão inteira de a por b .
%	Resto	Em $a\%b$ o resultado é o resto da divisão inteira de a por b , sendo que é um valor $0 \leq resto \leq b - 1$.
**	Potência	Em $a ** b$ o resultado de a^b .
abs	Módulo	Em $abs(a)$ o resultado é $-a$ se a é negativo e a se a é positivo.

Note a diferença entre as duas divisões

```
>>> 10/3
3.3333333333333335
>>> 10//3
3
>>> 10%3
1
```

3.4 Saída de Dados

Todo programa durante sua execução deve em princípio divulgar alguma coisa. Afinal para algo ele está sendo executado. Ainda que haja quase infinitas possíveis aplicações de programas, grande parte deles vai precisar dizer algo. Uma diferença importante entre Python 2 e Python 3 é que no 2, os parâmetros de print seguem o nome da função sem parênteses enquanto que os parênteses são obrigatórios no Python 3. Aliás este é um dos mecanismos para identificar se um programa originalmente foi escrito em 2 ou 3.

A maneira mais simples de obter o conteúdo de uma variável é apenas digitando seu nome seguido de ENTER. Veja

```
>>> a = 5
>>> b = a+2
>>> b
7
```

Uma maneira mais elaborada de saída é com o comando `print()`. É muito rico, com múltiplas opções, mas vamos começar lentamente. Na sua versão mais simples, o print admite uma lista de variáveis e eventualmente constantes a serem escritas na saída padrão quando o programa for executado. A saída padrão é um conceito importante em computação pessoal: trata-se do dispositivo que o computador usará para todas as saídas, quando nada for afirmado sobre ONDE fazer a saída. Em computação usa-se o conceito de *default* nestes casos. Então a saída *default* é conhecida como saída padrão e quase sempre é o monitor de vídeo. Vejam-se alguns exemplos

<code>print(a)</code>	Imprime o valor da variável a .
<code>print("o valor de a: ",a)</code>	Agora antes do valor da variável a , imprime-se a mensagem entre aspas. Ajuda a identificar as saídas
<code>print("a: ",a,"b: ",b)</code>	Imprime a e b identificando cada uma delas
<code>print("Desligue agora")</code>	Imprime apenas a mensagem
<code>print("Resultado ",x+22)</code>	Imprime a mensagem e depois o resultado de $x+22$. Note que o valor de x permanece inalterado

3.4.1 Opções da função print

Dentro do parênteses da função print, podem-se colocar algumas opções, a saber: `sep=` que indica qual o separador entre itens na função print. Por *default* é um espaço em branco, mas pode ser qualquer coisa. Veja os exemplos

```
>>> a=5
>>> b=6
>>> print(a,b)
5 6
>>> print(a,b,sep="#")
5#6
>>> print(a,b,sep="oba")
5oba6
>>>
```

Outra opção importante refere-se ao terminador da função. `end=` "n". Por default, é uma "nova linha", que em programação é reconhecida pelo caracter "

n". Mas, pode ser qualquer outra coisa. Chamar `print()` sem nenhum parâmetro, é o mesmo que chamar `print` com o parâmetro `end`. veja no exemplo

```
print("ufa",end="#")
ufa#
```

Finalmente, a terceira opção que vamos estudar é `file="nome"`. Ela indica para onde vai ser mandado o resultado de `print`. Se nada é dito, ele sempre vai para `sys.stdout` que é o nome oficial da saída padrão. Mas, o `print` pode ser mandado para qualquer lugar. Veja no exemplo

```
>>> saida=open("c:/p/python/saida","w")
>>> print("oba",file=saida)
>>> saida.close()
>>> saida=open("c:/p/python/saida","w")
>>> a
5
>>> print(a,file=saida)
>>> saida.close()
```

3.5 Entrada de dados

A grande maioria dos algoritmos (programas) precisam obter algum dado do operador antes de realizar o processamento. O comando básico aqui é `input`. Seu formato sempre será

```
variavel = input("mensagem")
```

O que acontece aqui:

- A mensagem citada no comando é impressa na saída padrão
- O programa é interrompido neste ponto
- depois que o usuário digitar alguma coisa e teclar **ENTER**, o que foi digitado é colocado na variável citada no comando
- O processamento segue sem mais interrupções.

Um detalhe importante é que os conteúdos SEMPRE serão carregados no formato de string. Sem transformações, estas variáveis não podem ser usadas, por exemplo, em computações numéricas. Para tanto elas deverão ser transformadas usando (entre outras) as funções: `int` converte algo para inteiro, `float` converte algo para flutuante, e já que estamos estudando conversões, tem-se também `str` que converte algo para string. Veja alguns exemplos

```
>>> idade=input("entre com a idade ")
entre com a idade 33
>>> idade
'33'
>>> idade=int(input("entre com a idade "))
entre com a idade 33
>>> idade
33
```

Perceba que no primeiro caso (sem a transformação `int`) a variável `idade` é do tipo string, o que se pode perceber nas aspas que são mostradas ('33'). No segundo caso, em que há a transformação de string para inteiro (representada pela chamada à função `int`), o conteúdo da variável é um número e portanto pode ser operada aritmeticamente.

3.5.1 Um programa completo usando `input` e `print`

Seja um programa para pedir o raio de um círculo e imprimir a área do mesmo usando a fórmula da geometria

$$S = \pi \times R^2$$

```
def area():
    a=float(input("Informe raio"))
    s=3.1415 * a
    print("a área é: ",s)
area()
```

Mais um exemplo

```

def prim2(n):
    import math
    if n==1:
        return False
    if n<4:
        return True
    if n%2==0:
        return False
    if n<9:
        return True
    if n%3==0:
        return False
    r=math.floor(math.sqrt(n))
    f=5
    while f<=r:
        if n%f==0:
            return False
        if (n%(f+2))==0:
            return False
        f=f+6
    return True

def enesprim(qual):
    count=1
    candidato=1
    while count<qual:
        candidato=candidato+2
        if prim2(candidato):
            count=count+1
    return(candidato)

aa=int(input("Informe qual a ordem do primo "))
bb=enesprim(aa)
print(bb)

```

Neste exemplo primeiro define-se a função `prim2` (bastante otimizada) que recebendo um valor qualquer informa se ele é ou não é primo. Depois define-se a função `enesprim` que recebe um número inteiro x e devolve o x -ésimo número primo. Finalmente, o script pede um número qualquer em `aa`, e com ele calcula o `aa`-ésimo número primo, e o joga em `bb`. Depois este número é impresso.

3.6 Variáveis Lógicas

Uma outra família de variáveis são as variáveis lógicas, geralmente obtidas como resultado de uma comparação ou pergunta. Essas variáveis só podem ter 2 valores que são Verdadeiro (em Python **True**) e Falso (**False**).

Usualmente, estas variáveis são obtidas ao operar condições. As principais condições em Python são

Símb	Nome	Como funciona
<code>==</code>	Igual	Em $a == b$, a resposta será True se a for igual a b e False em caso contrário.
<code>></code>	Maior	Em $a > b$ a resposta será True se a for maior do que b . Será False em caso contrário.
<code><</code>	Menor	Em $a < b$ a resposta será True se a for menor do que b . Será False em caso contrário.
<code>>=</code>	Maior ou igual	Em $a >= b$ a resposta será True se a for maior do que ou for igual a b . Será False em caso contrário.
<code><=</code>	Menor ou igual	Em $a <= b$ a resposta será True se a for menor do que ou igual a b . Será False em caso contrário.
<code>!=</code>	Diferente	Também conhecida como não igual, ela responde o contrário da operação igual <code>==</code> .

3.6.1 Expressões Lógicas

Quando há necessidade de expressar condições compostas, as operações acima podem ser conectadas usando-se 3 conectores lógicos que são:

Símb	Nome	Como funciona
and	E	Em $c1$ <i>and</i> $c2$, o resultado será verdadeiro se ambas, $c1$ e $c2$ forem verdadeiras e Falso, senão
or	OU	Em $c1$ <i>or</i> $c2$, o resultado será verdadeiro se $c1$ for verdadeiro, ou se $c2$ for verdadeiro ou se ambas forem verdadeiras. Será falso em caso contrário.
not	NÃO	Em <i>not</i> $c1$ o resultado será verdadeiro se $c1$ for falsa e será falsa se $c1$ for verdadeiro. Note que este operador é unário (só se aplica a uma condição).

Deve-se lembrar que estas operações e conectores estão definidas na matemática, particularmente na lógica de predicados. Lá seus símbolos são: \wedge para o conectivo E, \vee para o conectivo OU e finalmente o símbolo \sim para a negação lógica.

Capítulo 4

Comandos condicionais

Uma característica onipresente em linguagens de programação é a capacidade de executar ou não um conjunto de instruções a depender de uma variável lógica. Dizendo de outra maneira, pode-se comandar uma condição qualquer e se ela for verdadeira (que no Python é **True**) então um conjunto de instruções será executado. Se a condição for falsa (em Python **False**), ou nada é executado, ou também pode ocorrer, um segundo conjunto alternativo de instruções é executado.

O comando condicional por excelência em Python é **if** e ele têm o seguinte formato

```
if condição:
    comando_1
    comando_2
    ...
comando_n
```

Para escrever o comando, deve-se substituir a palavra condição por alguma condição relacional ou lógica, que ao final informe um valor **True** ou **False**. A regra é que os comandos 1, 2, ... só serão executados se a condição for **True**. Já ao final do bloco, o comando *n* será executado sempre independentemente da condição. Quem determina se o comando está ou não vinculado ao *if* é o espaçamento no início da linha, chamada em programação de indentação.

Em outras linguagens, usam-se caracteres delimitadores para os blocos de comando (**{** e **}** em C, Java e C++; *begin* e *end* em Pascal e Delphi), mas em Python consoante com a proposta de limpeza da linguagem, usam-se as margens para determinar as condições.

Como tudo na vida isto tem vantagens e desvantagens. Começando pelas primeiras, o código fica limpo e legível. Quem sempre exigia dos programadores que respeitassem as margens, agora ganha um aliado importante: se as margens forem bagunçadas o programa não funcionará bem. Além disso economizam-se caracteres dispensáveis.

Já a desvantagem decorre principalmente da inabilidade de muitos ambientes para manusear este tipo de arquivo. Muitos editores se arvoram em trocar um certo número de brancos por um ou mais caracteres de tabulação, e fazer isto é receita quase certa para problemas de operação dos programas. Além disso, muitos gerenciadores de download deixam de respeitar essas margens, fazendo com o que o recebedor de um programa fonte Python tenha sempre que revisar com cuidado o resultado de descargas de código.

Veja alguns exemplos

```
a = 5
if a==6:
    print('ufa')
print('agora')
```

Primeiro *a* recebe 5. Daí 5 é comparado para igual com 6. A resposta é **False**. Com isso, os comandos indentados abaixo não são executados (no caso `print('ufa')`). Finalmente é impresso 'agora' já que esta impressão independe do resultado da comparação.

Quem inicia um bloco indentado é a presença dos dois pontos : após a condição. Ele sinaliza ao editor em uso que a seguir deve-se respeitar a nova margem. Quando o programador quiser encerrar o bloco (e por conseguinte, a condição) ele terá que recuar manualmente o cursor até o início da linha.

Um segundo formato para o comando condicional, conhecido como condicional composta é aquele que apresenta uma condição e dois caminhos: um para o True e outro para o resultado False. Agora a separação entre um e outro é sinalizado pela palavra **else** que também deve ser seguida por dois pontos. Acompanhe no exemplo

```
a = 7
if a>3:
    print('oba')
else:
    print('ufa')
print('foi')
```

Acompanhe o raciocínio: Primeiro a é comparado com maior para 3. A resposta é `True` já que $7 > 3$ e com isso a impressão de 'oba' é feita. Como o resultado foi True, salta-se o bloco referente ao else:. Encerra o comando a impressão de 'foi' que independe do resultado da comparação. Veja agora um trecho parecido

```
a = 2
if a>3:
    print('oba')
else:
    print('ufa')
print('foi')
```

Agora a comparação resulta False (já que $2 > 3$ é falso) e com isso a impressão de 'oba' é saltada e depois a impressão de 'ufa' acontece. Novamente 'foi' é impresso independentemente da comparação inicial.

4.1 Exercício

Nos exercícios a seguir, você deve simular o interpretador Python e descobrir qual valor vai ser impresso ao final. Considere todas as variáveis como sendo do tipo **inteiro**. Todos os códigos estão sintaticamente corretos. A sugestão, que **depois** de ter calculado o valor na ponta do lápis, você confira o resultado com um computador real.

Exercício 1

```
def AA():
    A=4
    B=5
    C=8
    D=6
    F=6
    G=6
    if F!=2:
        C=A+F
        if (D>=9)or(C!=4):
            F=D+B
            C=C-C
    else:
        D=D-B
        if (D==5)and(D<8):
            D=D-B
        else:
            if F<2:
                F=A+B
            else:
                F=C+D
            A=G-G
            B=A*C
    A=C+D
    print(G+A+B-D)
AA()
```

Exercício 2

```
def AA():
    A=9
    B=5
    C=9
    D=2
    F=3
    G=8
    if B>5:
        if (G>9)or(D<=5):
            C=C+D
        if D==6:
            C=G-G
```

```

    C=G*A
else:
    G=G*C
    if (F<2)and(not(C<4)):
        A=G*F
        if C<=6:
            B=F*C
        else:
            if (C>5)or(not(G==3)):
                D=D+A
            else:
                A=F-B
            G=C-B
        B=G+D
    print(A+F+G-D)
AA()

```

Exercício 3

```

def AA():
    A=4
    B=7
    C=6
    D=2
    F=7
    G=9
    if C!=6:
        G=D+B
    if (not(G==4))and(not(B>5)):
        if B!=6:
            G=G-A
    G=G-C
    if (D!=6)or(not(A>4)):
        B=G*B
        if A<5:
            A=C*C
        else:
            G=C*B
    else:
        if (C!=9)or(B<=8):
            C=C*F
        else:
            A=B*F
            G=B*B
    C=B*C
    print(G+F+B-A)
AA()

```

Exercício 4

```

def AA():
    A=9
    B=7
    C=6
    D=9
    F=5
    G=6
    if A<6:
        C=F*A
    if (C<=7)and(A>6):
        if D==9:
            D=D+F
        else:

```

```

        if (F<3)or(not(F>2)):
            B=B*A
        else:
            if C>=2:
                G=C-B
            A=A+G
        G=F+F
    F=B-F
    if (not(B<=8))and(not(A<2)):
        G=G+A
        if A!=8:
            C=F-A
        else:
            G=F-B
    else:
        if (D!=8)and(not(C==4)):
            C=G+G
        else:
            A=D-A
        G=F+G
    C=G*D
    print(A+F+C-D)
AA()

```

Exercício 5

```

def AA():
    A=9
    B=7
    C=6
    D=3
    F=4
    G=7
    if G>4:
        if (B<=9)or(F<5):
            if D!=4:
                if (not(A<=5))or(A>4):
                    B=G+F
                else:
                    G=B-F
                G=A+B
            else:
                B=F-D
            G=G*D
        else:
            C=A+B
            F=G*C
    else:
        if G>2:
            if (A>=5)or(C!=3):
                if B<5:
                    G=F*B
                else:
                    D=F*G
            else:
                F=A+A
            F=C*C
        A=G+D
        print(D+F+G-A)
AA()

```

Exercício 6

```

def AA():
    A=9
    B=5
    C=5
    D=5
    F=6
    G=8
    if G>6:
        A=F-F
        if (D==2)and(not(G==9)):
            D=F+B
        B=B*D
    else:
        C=D+F
        if (G==8)or(not(D<7)):
            C=D+F
        else:
            if B<2:
                F=D-F
            else:
                F=B+B
            G=F*C
        B=F+D
    C=F+B
    print(C+A+F-B)
AA()

```

Exercício 7

```

def AA():
    A=4
    B=7
    C=7
    D=9
    F=2
    G=5
    if C>=5:
        if (A<=3)and(G!=2):
            F=A-G
        if A<=9:
            A=F*D
        A=C-A
    else:
        B=D*G
        if (D<5)or(A!=4):
            B=B-G
            if B>=5:
                F=D+G
        else:
            if (A==4)or(G>=4):
                D=C*B
            else:
                F=C*G
            F=F*G
        C=A+A
    print(G+A+C-D)
AA()

```

Exercício 8

```

def AA():
    A=4
    B=5

```

```
C=6
D=2
F=9
G=5
if C>6:
    G=C*B
if (F<5)and(A<6):
    if A>9:
        G=G-C
C=B+F
if (D>=2)or(C<=2):
    C=B+A
    if D>=7:
        B=B*D
    else:
        D=B*F
else:
    if (A<=2)or(G<4):
        D=F+A
    else:
        A=D*D
        A=F+B
A=G*G
print(B+C+D-A)
AA()
```

Respostas

1.	2.	3.	4.	5.	6.	7.	8.
11	5	-5	165	360	12	-8	34

Capítulo 5

Repetições

As repetições são estruturas da linguagem que asseguram a execução repetida de trechos de comandos, até que uma determinada condição for satisfeita. Note que este conceito faz parte do nosso dia a dia.

Por exemplo, uma caixa d'água posta a encher o fará até que atinja a capacidade total. Um forno assando um bolo deverá operar até que o bolo esteja assado, uma esteira carregadeira de um navio opera até que o depósito do navio esteja cheio ou enquanto o silo abastecedor tiver conteúdo, parando na condição que primeiro for atingida, um pneu de carro receberá ar comprimido até que uma certa pressão seja atingida e a água do café vai esquentar até ferver ou um pouco antes.

O que não falta na nossa vida são operações repetitivas que ocorrem até que uma condição seja atendida, ou o que praticamente é a mesma coisa, enquanto certa condição perdurar.

5.1 while

Os comandos básicos em Python são `while` que pode ser traduzido por enquanto. Seu formato é

```
while <condição>:
    comando_1
    comando_2
    ...
    comando_i
comando_n
```

Quando ele for encontrado será assim interpretado: primeiro a condição associada ao `while` será avaliada. Se ela for falsa, ocorre um desvio desprezando todo o bloco indentado abaixo. No exemplo acima, se a condição for falsa, o próximo comando a ser executado é o `comando_n`. Agora, se a condição é verdadeira, o bloco indentado é executado. Quando ele acabar, há um retorno (daí a repetição) ao início do bloco e a reavaliação da condição. Enquanto ela for verdadeira, o bloco fica em loop.

Logo, aqui vale uma advertência: quando usar um `while` garanta que em algum momento do bloco, a condição seja tornada falsa, sob pena do seu programa nunca mais sair daí.

Por exemplo, seja somar os 33 primeiros números ímpares:

```
i=1
q=0
p=1
while i<34:
    q=q+p
    p=p+2
    i=i+1
print(q)
```

O programa acima vai imprimir 1089, que é a soma procurada.

5.2 break

Um comando que permite modificar o funcionamento do `while` é o `break`. Normalmente ele é colocado dentro de uma condição, embora isto não seja obrigatório. Quando `break` é interpretado ele faz com que o ciclo do `while`, na verdade qualquer ciclo, seja interrompido, forçando uma saída do bloco indentado.

Exemplo, seja um programa para calcular o triplo de um número digitado. O programa deve operar até ser digitado um número negativo, que significará fim de processamento.

```

a=1
while 1==1:
    if a<0:
        break
    a=int(input("informe o valor "))
    print(3*a)
print('adeus')

```

Note que como vai-se usar o *break* para a saída do laço, escreve-se uma condição que sempre é verdadeira (conhecida na lógica como uma tautologia) , que neste caso é $1 == 1$, que, por óbvio, é sempre verdadeira.

Eis um trecho da execução desse programa:

```

===== RESTART: C:/Python3/teste.py =====
informe o valor 2
6
informe o valor 33
99
informe o valor 8
24
informe o valor -1
-3
adeus
>>>

```

5.2.1 Repetição aninhada

Quando se cria um bloco de repetição, dentro do bloco, o Python admite qualquer comando, inclusive um outro comando de repetição. Então surgem os blocos aninhados que são muito importantes em programação. Acompanhe o exemplo

```

i=1
while i<=3:
    j=10
    while j<=30:
        j=j+10
        print (i,j)
    i=i+1
print('acabou')

```

que vai gerar a seguinte saída

```

(1, 20)
(1, 30)
(1, 40)
(2, 20)
(2, 30)
(2, 40)
(3, 20)
(3, 30)
(3, 40)
acabou

```

5.3 Continue

Este comando também serve para interromper um ciclo de processamento, mas diferentemente de **break** que encerra inapelavelmente o laço, este comando **continue** encerra o processamento de uma interação e avança para a próxima interação. Acompanhe a diferença entre os dois comandos

```

i=3
while i<10:
    print(i,' ')
    if i%7==0:
        break
    i=i+1
print('acabou')

```

Esta execução teve o seguinte resultado:

```
3
4
5
6
7
acabou
```

Agora acompanhe o seguinte exemplo usando `continue`

```
i=3
while i<10:
    print(i,' ')
    if i%7==0:
        continue
    i=i+1
print('acabou')
```

Neste caso, não ponha para rodar, pois o programa vai entrar em laço infinito. Veja porque: O processamento começa com $i=3, 4, 5$ e 6 . Quando o valor 7 for atingido, o `continue` vai comandar um retorno à próxima interação, mas sem alterar o valor de i (já que esta alteração está depois do `continue`. Aí ela vai imprimir $7,7,7,7,\dots$ e não vai parar nunca mais.

5.4 For

O For é o grande interador do Python. Ele é uma expansão do comando `for` do C++. Lá no C++ o for funciona como se fosse uma simplificação do while, mas aqui ele é mais do que isso. A mudança se deve ao conceito de interador. Numa rápida explicação um interador é uma generalização do conceito de índice. Ele é uma generalização já que o indexador só se aplica a conjuntos homogêneos indexados (conhecidos como arrays) e o interador se aplica a qualquer coleção de objetos Python. Pode ser um array, mas pode ser também registros em um arquivo, itens em um conjunto, etc etc.

Veja um exemplo radical disso

```
s="Curitiba - Paraná - Brasil"
qtd=0
for c in s:
    if c=="a":
        qtd=qtd+1
print("achei ",qtd," a")
```

Ao executar o código acima, ele vai responder `achei 4 a`". Note que são 4 letras 'a', sendo que o 'a' acentuado de Paraná, não conta, já que é outro caracter.

5.5 Exercícios

Nos exercícios a seguir, você deve simular o interpretador Python e descobrir qual valor vai ser impresso ao final. Todos os códigos estão sintaticamente corretos. A sugestão é que **depois** de ter calculado o valor na ponta do lápis, você confira o resultado com um computador real.

Exemplos

Seja o programa

```
def AA():
    A=5
    B=5
    C=3
    if C>=18:
        while B<=18:
            C=C-3
            B=B+4
    else:
        C=C+4
        B=B+2
    print(C+C+B)
AA()
```

Cujo resultado é 21 e seja o seguinte código

```
def AA():
    A=4
    B=3
    C=6
    D=1
    C=C+6+3
    while C>6:
        B=D+7
        while A<=14:
            D=B+4
            A=A+2
        D=B+19
        C=C-1
    print(C+B+B)
AA()
```

cujo resultado é 438. Seja agora o programa

```
def AA():
    A=7
    B=3
    C=5
    D=6
    A=A+5+3
    while A>=4:
        B=D-4
        if A>=10:
            while C<=13:
                C=C+6
            B=D+3
        else:
            D=D-13
        if D>4:
            B=B+5
        else:
            B=D-5
        A=A-2
    print(B+B)
AA()
```

cujo resultado é -76.

Exercício 1

```
def AA():
    A=7
    B=6
    C=7
    while C<16:
        A=B+5
        B=A+5
        C=C+2
    print(A+B+C)
AA()
```

Informe no quadro próprio o valor que foi impresso.

Exercício 2

```
def AA():
    A=6
    B=2
    C=3
```

```

if C<=20:
    while B<=20:
        C=A+5
        B=B+4
    else:
        A=C-8
        B=B+2
print(C+A+B)
AA()

```

Exercício 3

```

def AA():
    A=8
    B=2
    C=2
    if B!=16:
        while C<16:
            B=B+8
            C=C+2
        else:
            while C<13:
                A=B+7
                C=C+2
    print(B+B+C)
AA()

```

Exercício 4

```

def AA():
    A=2
    B=1
    C=8
    D=6
    A=A+8+1
    while A>=5:
        C=D-5
        while B<=16:
            C=D-1
            B=B+6
        C=D-16
        A=A-1
    print(B+C+B)
AA()

```

Exercício 5

```

def AA():
    A=4
    B=2
    C=1
    D=5
    B=B+1+2
    while B>4:
        C=A+5
        if A>=9:
            while A<=12:
                C=C+2
                A=A+6
            C=D-16
        else:
            C=C-9
        B=B-4

```

```
print(B+C)
AA()
```

Exercício 6

```
def AA():
    A=3
    B=7
    C=8
    D=1
    A=A+8+7
    while A>5:
        B=B+3
        if B>=6:
            while D<=16:
                D=D+5
            B=C-7
        else:
            C=B+15
        if D>5:
            C=C+8
        else:
            B=C+10
        A=A-2
    print(B+C)
AA()
```

Exercício 7

```
def AA():
    A=6
    B=2
    C=1
    D=3
    C=C+1+2
    while C>=6:
        A=C+3
        if D>=9:
            while D<14:
                D=D+4
            B=B-3
        else:
            A=A+12
        if D<8:
            B=B+6
        else:
            A=A+9
        C=C-2
    print(A+B)
AA()
```

Exercício 8

```
def AA():
    A=5
    B=7
    C=8
    D=4
    D=D+8+7
    while D>=3:
        A=D+3
        if B>5:
            while B<=19:
```

```

        C=A+2
        B=B+1
        C=C-18
    else:
        A=C-10
        D=D-4
    print(C+D)
AA()

```

Exercício 9

```

def AA():
    A=8
    B=2
    C=7
    D=5
    D=D+7+2
    while D>7:
        A=B+1
        while B<=12:
            C=C-1
            B=B+1
            C=C+12
            D=D-1
        print(C+B+C)
AA()

```

Exercício 10

```

def AA():
    A=1
    B=2
    C=6
    if A<=15:
        while B<15:
            A=A-7
            B=B+4
    else:
        while B<16:
            A=C+1
            B=B+2
    print(A+A+B)
AA()

```

Respostas

1 124 39 132 28 1 119 8 -67 173 -36

Capítulo 6

Listas

Uma lista em Python é uma coleção de coisas. Em outras linguagens, essas coisas precisam ser de mesmo tipo, mas em Python, não. Podem ser coisas bem heterogêneas.

Cria-se uma lista em python usando-se os delimitadores colchetes. Uma lista é criada assim:

```
>>> a=[]
```

Criou-se aqui uma lista de nome `a`, inicialmente vazia. Para incluir coisas na lista, usa-se o método `append`, veja como

```
>>> a=[]
>>> a.append(1)
>>> a
[1]
>>> a.append('oi')
>>> a
[1, 'oi']
```

Outro jeito de criar uma lista é usando a função `list` como em

```
>>> b=list(range(6))
>>> b
[0, 1, 2, 3, 4, 5]
```

Note que o primeiro elemento da lista pode ser acessado para leitura fazendo-se

```
>>> a[0]
1
```

E podem ser alterados, como em

```
>>> a[0]='bacana'
>>> a
['bacana', 'oi']
```

Veja mais alguns exemplos, e estude-os...

```
>>> a = ['spam', 'eggs', 100, 1234]
>>> a
['spam', 'eggs', 100, 1234]
```

Os índices para as listas começam em 0 e crescem para a direita quando de usam números inteiros positivos. A função `len(lista)` devolve o tamanho da lista. Índices negativos vem da direita para a esquerda. `lista[-1]` devolve o último elemento da lista, `lista[-2]` o penúltimo e assim por diante. Veja isso tudo:

```
>>> x=[10,20,30,'oi','viva',4.5,99]
>>> x
[10, 20, 30, 'oi', 'viva', 4.5, 99]
>>> x[0]
10
>>> x[1]
20
>>> len(x)
7
>>> x[-1]
99
>>> x[-2]
4.5
```

6.1 Fatiamento

É uma operação fundamental em Python, pode ser usada em listas, tuplas, strings... É muito usada em MatLab, mas sinceramente eu não consigo dizer quem é o ovo e quem é a galinha, isto quem veio antes. Tanto faz. Uma fatia é uma especificação envolvendo um ou dois “dois pontos” e ela extrai um pedaço de uma lista, tupla ou string. A especificação é

`[começo da fatia:final da fatia:incremento]`

começo o começo da fatia é o índice do seu início, sempre relativo a zero. Se nada for escrito aqui o valor *default* é 0.

final o final da fatia é o próximo número ALÉM do final. Ou seja este número é compatível com o início da numeração em zero. O valor *default* é o tamanho da lista ou `len(lista)`.

incremento É um valor muitas vezes omitido (e neste caso o valor *default* é 1), mas quando presente ele indica quanto somar ao valor atual para obter o próximo valor.

Acompanhe alguns exemplos

```
>>> y=[11,21,31,41,51]
>>> y
[11, 21, 31, 41, 51]
>>> y[0:3]
[11, 21, 31]
>>> y[:3]
[11, 21, 31]
>>> y[2:]
[31, 41, 51]
>>> y[1:3:2]
[21]
>>> y[1:-1]
[21, 31, 41]
>>> y[::2]
[11, 31, 51]
>>> y[::-1]
[51, 41, 31, 21, 11]
>>> y[::-2]
[51, 31, 11]
>>> y[:3]
[11, 21, 31]
>>> y[3:]
[41, 51]
```

Da mesma forma que índices de string, índices de lista começam em 0, listas também podem ser concatenadas, fatiadas e multiplicadas:

```
>>> a = ['spam', 'eggs', 100, 1234]
>>> a[0]
'spam'
>>> a[3]
1234
>>> a[-2]
100
>>> a[1:-1]
['eggs', 100]
>>> a[:2] + ['bacon', 2*2]
['spam', 'eggs', 'bacon', 4]
>>> 3*a[:3] + ['Boo!']
['spam', 'eggs', 100, 'spam', 'eggs', 100, 'spam', 'eggs', 100, 'Boo!']
```

Todas as operações de fatiamento devolvem uma nova lista contendo os elementos solicitados. Isto significa que o fatiamento a seguir retorna uma cópia rasa (shallow copy) da lista:

```
>>> a[:]
['spam', 'eggs', 100, 1234]
```

Diferentemente de strings, que são imutáveis, é possível alterar elementos individuais de uma lista:

```

>>>
>>> a
['spam', 'eggs', 100, 1234]
>>> a[2] = a[2] + 23
>>> a
['spam', 'eggs', 123, 1234]

```

Atribuição à fatias também é possível, e isso pode até alterar o tamanho da lista ou remover todos os itens dela:

```

>>>
>>> # Substituir alguns itens:
... a[0:2] = [1, 12]
>>> a
[1, 12, 123, 1234]
>>> # Remover alguns:
... a[0:2] = []
>>> a
[123, 1234]
>>> # Inserir alguns:
... a[1:1] = ['bletch', 'xyzyy']
>>> a
[123, 'bletch', 'xyzyy', 1234]
>>> # Inserir uma cópia da própria lista no início
>>> a[:0] = a
>>> a
[123, 'bletch', 'xyzyy', 1234, 123, 'bletch', 'xyzyy', 1234]
>>> # Limpar a lista: substituir todos os itens por uma lista vazia
>>> a[:] = []
>>> a
[]

```

A função embutida `len()` também se aplica a listas:

```

>>>
>>> a = ['a', 'b', 'c', 'd']
>>> len(a)
4

```

É possível aninhar listas (criar listas contendo outras listas), por exemplo:

```

>>>
>>> q = [2, 3]
>>> p = [1, q, 4]
>>> len(p)
3
>>> p[1]
[2, 3]
>>> p[1][0]
2
>>> p[1].append('xtra')      # Veja a seção 5.1
>>> p
[1, [2, 3, 'xtra'], 4]
>>> q
[2, 3, 'xtra']

```

Observe que no último exemplo, `p[1]` e `q` na verdade se referem ao mesmo objeto! Mais tarde retornaremos a semântica dos objetos.

6.1.1 fatiamento em arrays

Acompanhe

```

>>> from numpy import *
>>> a=array([[1,2,3],[4,5,6],[7,8,9],[10,11,12]])
>>> a
array([[ 1,  2,  3],

```

```

    [ 4,  5,  6],
    [ 7,  8,  9],
    [10, 11, 12]])
>>> a[1,1]
5
>>> a[1:3,1]
array([5, 8])
>>> a[1:3,1:3]
array([[5, 6],
       [8, 9]])
>>> a[1:3,1:3]=777
>>> a
array([[ 1,  2,  3],
       [ 4, 777, 777],
       [ 7, 777, 777],
       [10, 11, 12]])

```

6.2 Mais

Este texto descreve alguns pontos já abordados, porém com mais detalhes, e adiciona outros pontos e foi retirado de <http://turing.com.br/pydoc/2.7/tutorial/datastructures.html>

6.3 Mais sobre listas

O tipo list possui mais métodos. Aqui estão todos os métodos disponíveis em um objeto lista:

`list.append(x)`

Adiciona um item ao fim da lista; equivale a `a[len(a):] = [x]`.

`list.extend(L)`

Prolonga a lista, adicionando no fim todos os elementos da lista L passada como argumento; equivalente a `a a[len(a):] = L`.

`list.insert(i, x)`

Inserir um item em uma posição especificada. O primeiro argumento é o índice do elemento antes do qual será feita a inserção, assim `a.insert(0, x)` insere no início da lista, e `a.insert(len(a), x)` equivale a `a.append(x)`.

`list.remove(x)`

Remove o primeiro item encontrado na lista cujo valor é igual a x. Se não existir valor igual, uma exceção `ValueError` é levantada.

`list.pop([i])`

Remove o item na posição dada e o devolve. Se nenhum índice for especificado, `a.pop()` remove e devolve o último item na lista. (Os colchetes ao redor do i indicam que o parâmetro é opcional, não que você deva digitá-los daquela maneira. Você verá essa notação com frequência na Referência da Biblioteca Python.)

`list.index(x)`

Devolve o índice do primeiro item cujo valor é igual a x, gerando `ValueError` se este valor não existe

`list.count(x)`

Devolve o número de vezes que o valor x aparece na lista.

`list.sort()`

Ordena os itens na própria lista in place.

`list.reverse()`

Inverte a ordem dos elementos na lista in place (sem gerar uma nova lista).

Um exemplo que utiliza a maioria dos métodos::

```

>>> a = [66.25, 333, 333, 1, 1234.5]
>>> print a.count(333), a.count(66.25), a.count('x')
2 1 0
>>> a.insert(2, -1)
>>> a.append(333)
>>> a
[66.25, 333, -1, 333, 1, 1234.5, 333]
>>> a.index(333)
1
>>> a.remove(333)
>>> a
[66.25, -1, 333, 1, 1234.5, 333]
>>> a.reverse()
>>> a
[333, 1234.5, 1, 333, -1, 66.25]
>>> a.sort()
>>> a
[-1, 1, 66.25, 333, 333, 1234.5]

```

(N.d.T. Note que os métodos que alteram a lista, inclusive `sort` e `reverse`, devolvem `None` para lembrar o programador de que modificam a própria lista, e não criam uma nova. O único método que altera a lista e devolve um valor é o `pop`)

6.3.1 Usando listas como pilhas

Os métodos de lista tornam muito fácil utilizar listas como pilhas, onde o item adicionado por último é o primeiro a ser recuperado (política “último a entrar, primeiro a sair”). Para adicionar um item ao topo da pilha, use `append()`. Para recuperar um item do topo da pilha use `pop()` sem nenhum índice. Por exemplo:

```

>>> pilha = [3, 4, 5]
>>> pilha.append(6)
>>> pilha.append(7)
>>> pilha
[3, 4, 5, 6, 7]
>>> pilha.pop()
7
>>> pilha
[3, 4, 5, 6]
>>> pilha.pop()
6
>>> pilha.pop()
5
>>> pilha
[3, 4]

```

6.3.2 Usando listas como filas

Você também pode usar uma lista como uma fila, onde o primeiro item adicionado é o primeiro a ser recuperado (política “primeiro a entrar, primeiro a sair”); porém, listas não são eficientes para esta finalidade. Embora `appends` e `pops` no final da lista sejam rápidos, fazer `inserts` ou `pops` no início da lista é lento (porque todos os demais elementos têm que ser deslocados).

Para implementar uma fila, use a classe `collections.deque` que foi projetada para permitir `appends` e `pops` eficientes nas duas extremidades. Por exemplo:

```

>>> from collections import deque
>>> fila = deque(["Eric", "John", "Michael"])
>>> fila.append("Terry")      # Terry chega
>>> fila.append("Graham")    # Graham chega
>>> fila.popleft()           # O primeiro a chegar parte
'Eric'
>>> fila.popleft()           # O segundo a chegar parte
'John'
>>> fila                      # O resto da fila, em ordem de chegada
deque(['Michael', 'Terry', 'Graham'])
(N.d.T. neste exemplo são usados nomes de membros do grupo Monty Python)

```

6.4 Ferramentas de programação funcional

Existem três funções embutidas que são muito úteis para processar listas: `filter()`, `map()`, e `reduce()`.

`filter(função, sequência)` devolve uma nova sequência formada pelos itens do segundo argumento para os quais `funcao(item)` é verdadeiro. Se a sequência de entrada for string ou tupla, a saída será do mesmo tipo; caso contrário, o resultado será sempre uma lista. Por exemplo, para computar uma sequência de números não divisíveis por 2 ou 3:

```
>>> def f(x): return x % 2 != 0 and x % 3 != 0
...
>>> filter(f, range(2, 25))
[5, 7, 11, 13, 17, 19, 23]
map(funcao, sequencia) aplica funcao(item) a cada item da sequência e devolve uma lista formada
...
>>> def cubo(x): return x*x*x
...
>>> map(cubo, range(1, 11))
[1, 8, 27, 64, 125, 216, 343, 512, 729, 1000]
```

Mais de uma sequência pode ser passada; a função a ser aplicada deve aceitar tantos argumentos quantas sequências forem passadas, e é invocada com o item correspondente de cada sequência (ou `None`, se alguma sequência for menor que outra). Por exemplo:

```
>>> seq = range(8)
>>> def somar(x, y): return x+y
...
>>> map(somar, seq, seq)
[0, 2, 4, 6, 8, 10, 12, 14]
```

Se `None` for passado no lugar da função, então será aplicada a função identidade (apenas devolve o argumento recebido). Se várias sequências forem passadas, a lista resultante terá tuplas formadas pelos elementos correspondentes de cada sequência. Isso se parece com a função `zip()`, exceto que `map()` devolve uma lista com o comprimento da sequência mais longa que foi passada, preenchendo as lacunas com `None` quando necessário, e `zip()` devolve uma lista com o comprimento da mais curta. Confira:

```
>>> map(None, range(5))
[0, 1, 2, 3, 4]
>>> map(None, range(5), range(3))
[(0, 0), (1, 1), (2, 2), (3, None), (4, None)]
>>> zip(range(5), range(3))
[(0, 0), (1, 1), (2, 2)]
```

A função `reduce(função, sequência)` devolve um único valor construído a partir da sucessiva aplicação da função binária (N.d.T. que recebe dois argumentos) a todos os elementos da lista fornecida, começando pelos dois primeiros itens, depois aplicando a função ao primeiro resultado obtido e ao próximo item, e assim por diante. Por exemplo, para computar a soma dos inteiros de 1 a 10:

```
>>> def somar(x,y): return x+y
...
>>> reduce(somar, range(1, 11))
55
```

Se houver um único elemento na sequência fornecida, seu valor será devolvido. Se a sequência estiver vazia, uma exceção será levantada.

Um terceiro argumento pode ser passado para definir o valor inicial. Neste caso, redução de uma sequência vazia devolve o valor inicial. Do contrário, a redução se inicia aplicando a função ao valor inicial e ao primeiro elemento da sequência, e continuando a partir daí.

```
>>> def somatoria(seq):
...     def somar(x,y): return x+y
...     return reduce(somar, seq, 0)
...
>>> somatoria(range(1, 11))
55
>>> somatoria([])
0
```

Não use a função `somatoria` deste exemplo; somar sequências de números é uma necessidade comum, e para isso Python tem a função embutida `sum()`, que faz exatamente isto, e também aceita um valor inicial (opcional).

6.5 List comprehensions ou abrangências de listas

Uma list comprehension é uma maneira concisa de construir uma lista preenchida. (N.d.T. literalmente, abrangência de lista mas no Brasil o termo em inglês é muito usado; também se usa a abreviação listcomp)

Um uso comum é construir uma nova lista onde cada elemento é o resultado de alguma expressão aplicada a cada membro de outra sequência ou iterável, ou para construir uma subsequência cujos elementos satisfazem uma certa condição.

Por exemplo, suponha que queremos criar uma lista de quadrados, assim:

```
>>> quadrados = []
>>> for x in range(10):
...     quadrados.append(x**2)
...
>>> quadrados
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
Podemos obter o mesmo resultado desta forma:
```

```
quadrados = [x**2 for x in range(10)]
```

Isso equivale a `quadrados = map(lambda x: x**2, range(10))`, mas é mais conciso e legível.

Uma abrangência de lista é formada por um par de colchetes contendo uma expressão seguida de uma cláusula for, e então zero ou mais cláusulas for ou if. O resultado será uma lista resultante da avaliação da expressão no contexto das cláusulas for e if.

Por exemplo, esta listcomp combina os elementos de duas listas quando eles são diferentes:

```
>>> [(x, y) for x in [1,2,3] for y in [3,1,4] if x != y]
[(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]
Isto equivale a:
```

```
>>> combs = []
>>> for x in [1,2,3]:
...     for y in [3,1,4]:
...         if x != y:
...             combs.append((x, y))
...
>>> combs
[(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]
```

Note como a ordem dos for e if é a mesma nos dois exemplos acima.

Se a expressão é uma tupla, ela deve ser inserida entre parênteses (ex., (x, y) no exemplo anterior).

```
>>> vec = [-4, -2, 0, 2, 4]
>>> # criar uma lista com os valores dobrados
>>> [x*2 for x in vec]
[-8, -4, 0, 4, 8]
>>> # filtrar a lista para excluir números negativos
>>> [x for x in vec if x >= 0]
[0, 2, 4]
>>> # aplicar uma função a todos os elementos
>>> [abs(x) for x in vec]
[4, 2, 0, 2, 4]
>>> # invocar um método em cada elemento
>>> frutas = [' banana', ' loganberry ', 'passion fruit ']
>>> [arma.strip() for arma in frutas]
['banana', 'loganberry', 'passion fruit']
>>> # criar uma lista de duplas, ou tuplas de 2, como (numero, quadrado)
>>> [(x, x**2) for x in range(6)]
[(0, 0), (1, 1), (2, 4), (3, 9), (4, 16), (5, 25)]
>>> # a tupla deve estar entre parênteses, do contrário ocorre um erro
>>> [x, x**2 for x in range(6)]
File "<stdin>", line 1
    [x, x**2 for x in range(6)]
        ^
SyntaxError: invalid syntax
```

```
>>> # achatar uma lista usando uma listcomp com dois 'for'
```

```
>>> vec = [[1,2,3], [4,5,6], [7,8,9]]
>>> [num for elem in vec for num in elem]
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

A abrangência de lista é mais flexível do que `map()` e pode conter expressões complexas e funções aninhadas, sem necessidade do uso de `lambda`:

```
>>> from math import pi
>>> [str(round(pi, i)) for i in range(1, 6)]
['3.1', '3.14', '3.142', '3.1416', '3.14159']
```

6.5.1 Listcomps aninhadas

A expressão inicial de uma listcomp pode ser uma expressão arbitrária, inclusive outra listcomp.

Observe este exemplo de uma matriz 3x4 implementada como uma lista de 3 listas de comprimento 4:

```
>>> matriz = [
...     [1, 2, 3, 4],
...     [5, 6, 7, 8],
...     [9, 10, 11, 12],
... ]
```

A abrangência de listas abaixo transpõe as linhas e colunas:

```
>>> [[linha[i] for linha in matriz] for i in range(len(matriz[0]))]
[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]
```

Como vimos na seção anterior, a listcomp aninhada é computada no contexto da cláusula `for` seguinte, portanto o exemplo acima equivale a:

```
>>> transposta = []
>>> for i in range(len(matriz[0])):
...     transposta.append([linha[i] for linha in matriz])
...
>>> transposta
[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]
e isso, por sua vez, faz o mesmo que isto:
```

```
>>> transposta = []
>>> for i in range(len(matriz[0])):
...     # as próximas 3 linhas implementam a listcomp aninhada
...     linha_transposta = []
...     for linha in matriz:
...         linha_transposta.append(linha[i])
...     transposta.append(linha_transposta)
...
>>> transposta
[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]
```

Na prática, você deve dar preferência a funções embutidas em vez de expressões complexas. A função `zip()` resolve muito bem este caso de uso:

6.6 O comando `del`

Existe uma maneira de remover um item de uma lista conhecendo seu índice, ao invés de seu valor: o comando `del`. Ele difere do método `list.pop()`, que devolve o item removido. O comando `del` também pode ser utilizado para remover fatias (slices) da lista, ou mesmo limpar a lista toda (que fizemos antes atribuindo uma lista vazia à fatia `a[:]`). Por exemplo:

```
>>> a = [-1, 1, 66.25, 333, 333, 1234.5]
>>> del a[0]
>>> a
[1, 66.25, 333, 333, 1234.5]
>>> del a[2:4]
>>> a
[1, 66.25, 1234.5]
>>> del a[:]
```

```
>>> a
[]
```

del também pode ser usado para remover totalmente uma variável:

```
>>> del a
>>> a
Traceback (most recent call last):
...
NameError: name 'a' is not defined
```

Referenciar a variável a depois de sua remoção constitui erro (pelo menos até que seja feita uma nova atribuição para ela). Encontraremos outros usos para o comando del mais tarde.

6.7 Tuplas e seqüências

Vimos que listas e strings têm muitas propriedades em comum, como indexação e operações de fatiamento (slicing). Elas são dois exemplos de seqüências (veja Sequence Types — str, unicode, list, tuple, bytearray, buffer, xrange). Como Python é uma linguagem em evolução, outros tipos de seqüências podem ser adicionados. Existe ainda um outro tipo de seqüência padrão na linguagem: a tupla (tuple).

Uma tupla consiste em uma seqüência de valores separados por vírgulas:

```
>>> t = 12345, 54321, 'bom dia!'
>>> t[0]
12345
>>> t
(12345, 54321, 'bom dia!')
>>> # Tuplas podem ser aninhadas:
... u = t, (1, 2, 3, 4, 5)
>>> u
((12345, 54321, 'bom dia!'), (1, 2, 3, 4, 5))
```

Como você pode ver no trecho acima, na saída do console as tuplas são sempre envolvidas por parênteses, assim tuplas aninhadas podem ser lidas corretamente. Na criação, tuplas podem ser envolvidas ou não por parênteses, desde que o contexto não exija os parênteses (como no caso da tupla dentro a uma expressão maior).

Tuplas podem ser usadas de diversas formas: pares ordenados (x, y), registros de funcionário extraídos uma base de dados, etc. Tuplas, assim como strings, são imutáveis: não é possível atribuir valores a itens individuais de uma tupla (você pode simular o mesmo efeito através de operações de fatiamento e concatenação; N.d.T. mas neste caso nunca estará modificando tuplas, apenas criando novas). Também é possível criar tuplas contendo objetos mutáveis, como listas.

Um problema especial é a criação de tuplas contendo 0 ou 1 itens: a sintaxe usa certos truques para acomodar estes casos. Tuplas vazias são construídas por uma par de parênteses vazios; uma tupla unitária é construída por um único valor e uma vírgula entre parênteses (não basta colocar um único valor entre parênteses). Feio, mas funciona:

```
>>> vazia = ()
>>> upla = 'hello', # <-- note a vírgula no final
>>> len(vazia)
0
>>> len(upla)
1
>>> upla
('hello',)
```

O comando `t = 12345, 54321, 'hello!'` é um exemplo de empacotamento de tupla (tuple packing): os valores 12345, 54321 e 'bom dia!' são empacotados juntos em uma tupla. A operação inversa também é possível:

```
>>> x, y, z = t
```

Isto é chamado de desempacotamento de seqüência (sequence unpacking), funciona para qualquer tipo de seqüência do lado direito. Para funcionar, é necessário que a lista de variáveis do lado esquerdo tenha o mesmo comprimento da seqüência à direita. Sendo assim, a atribuição múltipla é um caso de empacotamento de tupla e desempacotamento de seqüência:

6.8 Sets (conjuntos)

Python também inclui um tipo de dados para conjuntos, chamado set. Um conjunto é uma coleção desordenada de elementos, sem elementos repetidos. Usos comuns para sets incluem a verificação eficiente da existência de objetos e a eliminação de itens duplicados. Conjuntos também suportam operações matemáticas como união, interseção, diferença e diferença simétrica.

Uma pequena demonstração:

```
>>> cesta = ['uva', 'laranja', 'uva', 'abacaxi', 'laranja', 'banana']
>>> frutas = set(cesta) # criar um conjunto sem duplicatas
>>> frutas
set(['abacaxi', 'uva', 'laranja', 'banana'])
>>> 'laranja' in frutas # testar se um elemento existe é muito rápido
True
>>> 'capim' in frutas
False
>>>
>>> # Demonstrar operações de conjunto em letras únicas de duas palavras
...
>>> a = set('abracadabra')
>>> b = set('alacazam')
>>> a # letras unicas em a
set(['a', 'r', 'b', 'c', 'd'])
>>> a - b # letras em a mas não em b
set(['r', 'd', 'b'])
>>> a | b # letras em a ou em b
set(['a', 'c', 'r', 'd', 'b', 'm', 'z', 'l'])
>>> a & b # letras tanto em a como em b
set(['a', 'c'])
>>> a ^ b # letras em a ou b mas não em ambos
set(['r', 'd', 'b', 'm', 'z', 'l'])
```

N.d.T. A sintaxe de sets do Python 3.1 foi portada para o Python 2.7, tornando possível escrever 10, 20, 30 para definir set([10, 20, 30]). O conjunto vazio tem que ser escrito como set() ou set([]), pois sempre representou um dicionário vazio, como veremos a seguir. Também existe uma sintaxe para set comprehensions, que nos permite escrever x*10 for x in [1, 2, 3] para construir 10, 20, 30.

6.9 Dicionários

Outra estrutura de dados muito útil embutida em Python é o dicionário, cujo tipo é dict (ver Mapping Types — dict). Dicionários são também chamados de “memória associativa” ou “vetor associativo” em outras linguagens. Diferente de seqüências que são indexadas por inteiros, dicionários são indexados por chaves (keys), que podem ser de qualquer tipo imutável (como strings e inteiros). Tuplas também podem ser chaves se contiverem apenas strings, inteiros ou outras tuplas. Se a tupla contiver, direta ou indiretamente, qualquer valor mutável, não poderá ser chave. Listas não podem ser usadas como chaves porque podem ser modificadas in place pela atribuição em índices ou fatias, e por métodos como append() e extend().

Um bom modelo mental é imaginar um dicionário como um conjunto não ordenado de pares chave-valor, onde as chaves são únicas em uma dada instância do dicionário. Dicionários são delimitados por chaves: {}, e contém uma lista de pares chave:valor separada por vírgulas. Dessa forma também será exibido o conteúdo de um dicionário no console do Python. O dicionário vazio é {}.

As principais operações em um dicionário são armazenar e recuperar valores a partir de chaves. Também é possível remover um par chave:valor com o comando del. Se você armazenar um valor utilizando uma chave já presente, o antigo valor será substituído pelo novo. Se tentar recuperar um valor usando uma chave inexistente, será gerado um erro.

O método keys() do dicionário devolve a lista de todas as chaves presentes no dicionário, em ordem arbitrária (se desejar ordená-las basta aplicar o a função sorted() à lista devolvida). Para verificar a existência de uma chave, use o operador in.

A seguir, um exemplo de uso do dicionário:

```
>>> d={}
>>> d
{}
>>> d[123]='oba' #a chave 123 corresponde a 'oba'
>>> d
```

```

{123: 'oba'}
>>> d[124]='fui' #a chave 124 a 'fui'
>>> d
{123: 'oba', 124: 'fui'}

>>> tel = {'jack': 4098, 'sape': 4139}
>>> tel['guido'] = 4127
>>> tel
{'sape': 4139, 'guido': 4127, 'jack': 4098}
>>> tel['jack']
4098
>>> del tel['sape']
>>> tel['irv'] = 4127
>>> tel
{'guido': 4127, 'irv': 4127, 'jack': 4098}
>>> tel.keys()
['guido', 'irv', 'jack']
>>> 'guido' in tel
True

```

O construtor `dict()` produz dicionários diretamente a partir de uma lista de chaves-valores, armazenadas como duplas (tuplas de 2 elementos). Quando os pares formam um padrão, uma list comprehensions pode especificar a lista de chaves-valores de forma mais compacta.

```

>>>
>>> dict([('sape', 4139), ('guido', 4127), ('jack', 4098)])
{'sape': 4139, 'jack': 4098, 'guido': 4127}
>>> dict([(x, x**2) for x in (2, 4, 6)]) # use uma list comprehension
{2: 4, 4: 16, 6: 36}

```

N.d.T. A partir do Python 2.7 também existem dict comprehensions (abrangências de dicionário). Com esta sintaxe o último dict acima pode ser construído assim `x: x**2 for x in (2, 4, 6)`.

Mais adiante no tutorial aprenderemos sobre expressões geradoras, que são ainda mais adequados para fornecer os pares de chave-valor para o construtor `dict()`.

Quando chaves são strings simples, é mais fácil especificar os pares usando argumentos nomeados no construtor:

```

>>> dict(sape=4139, guido=4127, jack=4098)
{'sape': 4139, 'jack': 4098, 'guido': 4127}

```

N.d.T. Naturalmente, por limitações da sintaxe, essa sugestão só vale se as chaves forem strings ASCII, sem acentos, conforme a regras para formação de identificadores do Python.

6.10 Técnicas de iteração

Ao percorrer um dicionário em um laço, a variável de iteração receberá uma chave de cada vez:

```

>>> knights = {'gallahad': 'the pure', 'robin': 'the brave'}
>>> for k in knights:
...     print k
...
gallahad
robin

```

Quando conveniente, a chave e o valor correspondente podem ser obtidos simultaneamente com o método `iteritems()`.

```

>>> knights = {'gallahad': 'the pure', 'robin': 'the brave'}
>>> for k, v in knights.iteritems():
...     print k, v
...
gallahad the pure
robin the brave

```

Ao percorrer uma sequência qualquer, o índice da posição atual e o valor correspondente podem ser obtidos simultaneamente usando a função `enumerate()`:

```
>>> for i, v in enumerate(['tic', 'tac', 'toe']):
...     print i, v
...
0 tic
1 tac
2 toe
```

Para percorrer duas ou mais sequências simultaneamente com o laço, os itens podem ser agrupados com a função `zip()`.

```
>>> questions = ['name', 'quest', 'favorite color']
>>> answers = ['lancelot', 'the holy grail', 'blue']
>>> for q, a in zip(questions, answers):
...     print 'What is your {0}? It is {1}.'.format(q, a)
...
...

```

What is your name? It is lancelot. What is your quest? It is the holy grail. What is your favorite color? It is blue. N.d.T. O exemplo acima reproduz um diálogo do filme Monty Python - Em Busca do Cálice Sagrado. Este trecho não pode ser traduzido, exceto por um decreto real.

Para percorrer uma sequência em ordem inversa, chame a função `reversed()` com a sequência na ordem original.

```
>>> for i in reversed(xrange(1,10,2)):
...     print i
...
9
7
5
3
1
```

Para percorrer uma sequência de maneira ordenada, use a função `sorted()`, que retorna uma lista ordenada com os itens, mantendo a sequência original inalterada.

```
>>> cesta = ['uva', 'laranja', 'uva', 'abacaxi', 'laranja', 'banana']
>>> for fruta in sorted(cesta):
...     print fruta
...
abacaxi
banana
laranja
laranja
uva
uva
>>> for fruta in sorted(set(cesta)): # sem duplicações
...     print fruta
...
abacaxi
banana
laranja
uva
```

6.10.1 Métodos de dicionários

`.clear()`

Remove todos os itens do dicionário

```
>>> d={'chave':1, 'chave2': 2}
>>> d
{'chave': 1, 'chave2': 2}
>>> d.clear()
>>> d
{}
```

`.get()`

Retorna o valor associado à chave no dicionário. Se a chave não existe retorna **valor**. Se **valor** não é usado o padrão é **None**.

```
>>> d={'arroz':20, 'feijão':30}
>>> d.get('ovo')
>>> print(d.get('ovo'))
None
>>> print(d.get('ovo',33))
33
```

`.keys()`

Retorna uma lista com todas as chaves do dicionário.

6.11 Mais sobre condições

As condições de controle usadas em `while` e `if` podem conter quaisquer operadores, não apenas comparações.

Os operadores de comparação `in` e `not in` verificam se um valor ocorre (ou não ocorre) em uma dada sequência. Os operadores `is` e `is not` comparam se dois objetos são na verdade o mesmo objeto; isto só é relevante no contexto de objetos mutáveis, como listas. Todos os operadores de comparação possuem a mesma precedência, que é menor do que a prioridade de todos os operadores numéricos.

Comparações podem ser encadeadas: Por exemplo `a < b == c` testa se `a` é menor que `b` e também se `b` é igual a `c`.

Comparações podem ser combinadas através de operadores booleanos `and` e `or`, e o resultado de uma comparação (ou de qualquer outra expressão), pode ter seu valor booleano negado através de `not`. Estes possuem menor prioridade que os demais operadores de comparação. Entre eles, `not` é o de maior prioridade e `or` o de menor. Dessa forma, a condição `A and not B or C` é equivalente a `(A and (not B)) or C`. Naturalmente, parênteses podem ser usados para expressar o agrupamento desejado.

Os operadores booleanos `and` e `or` são operadores short-circuit: seus argumentos são avaliados da esquerda para a direita, e a avaliação para quando o resultado é determinado. Por exemplo, se `A` e `C` são expressões verdadeiras, mas `B` é falsa, então `A and B and C` não chega a avaliar a expressão `C`. Em geral, quando usado sobre valores genéricos e não como booleanos, o valor do resultado de um operador atalho é o último valor avaliado na expressão.

É possível atribuir o resultado de uma comparação ou outra expressão booleana para uma variável. Por exemplo:

```
>>> string1, string2, string3 = '', 'Trondheim', 'Hammer Dance'
>>> non_null = string1 or string2 or string3
>>> non_null
'Trondheim'
```

Observe que em Python, diferente de C, atribuição não pode ocorrer dentro de uma expressão. Programadores C podem resmungar, mas isso evita toda uma classe de problemas frequentemente encontrados em programas C: digitar `=` numa expressão quando a intenção era `==`.

6.12 Brincando com listas

Criando uma lista de números em sequência:

```
# python 2:
lista = range(100)
```

```
# python 3:
lista = list(range(100))
```

Criando uma lista com list comprehensions:

```
>>> lista = [x*2 for x in range(100)]
```

Percorrendo uma lista com `for in`:

```
>>> for x in lista:
    # faça algo com x
```

Percorrendo uma lista, obtendo os valores e seus índices:

```
>>> for indice, valor in enumerate(lista):
    print "lista[%d] = %d" % (indice, valor)
```

Percorrendo um pedaço de uma lista usando slicing:

```
>>> for x in lista[40:60]:
    # faça algo com x
```

Percorrendo uma lista de trás pra frente definindo o passo do slicing como -1:

```
>>> for x in lista[::-1]:  
    # faça algo com x
```

Ou:

```
for x in reversed(lista):  
    # faça algo com x
```

Percorrendo uma lista ordenada:

```
>>> for x in sorted(lista):  
    # faça algo com x
```

Acessando o último elemento de uma lista com o índice -1:

```
>>> print lista[-1]
```

Copiando uma referência para uma lista:

```
>>> nova_ref = lista  
>>> nova_ref is lista  
True
```

Copiando de verdade uma lista:

```
>>> nova_lista = lista[:]  
>>> nova_lista is lista  
False  
Ou, usando o módulo copy:  
>>> import copy  
>>> nova_lista = copy.copy(lista)  
>>> nova_lista is lista  
False
```

Ou caso lista contivesse listas aninhadas e quiséssemos fazer com que estas também fossem completamente copiadas, e não somente referenciadas, usaríamos a função `deepcopy()`:

```
>>> nova_lista = copy.deepcopy(lista)  
>>> nova_lista is lista  
False
```

Embaralhando os elementos de uma lista (in-place):

```
>>> import random  
>>> random.shuffle(lista) # altera a própria lista
```

Obtendo uma amostra aleatória (de 10 elementos) de uma lista:

```
>>> print random.sample(lista, 10)  
[729, 9025, 2401, 8100, 5776, 784, 1444, 484, 6241, 7396]
```

Obtendo um elemento aleatório de uma lista:

```
>>> random.choice(lista)  
7744
```

Gerando uma lista com 10 números aleatórios, com valores entre 0 e 99:

```
>>> lista_aleatoria = random.sample(range(0, 100), 10)
```

Obtendo o maior elemento de uma lista:

```
>>> lista = range(0, 10)  
>>> print max(lista)  
9  
O menor:  
>>> print min(lista)  
0
```

Pegando somente os elementos de índice par:

```
>>> print lista[::2]
[0, 2, 4, 6, 8]
Os de índice ímpar:
>>> print lista[1::2]
[1, 3, 5, 7, 9]
```

Somando todos os elementos de uma lista:

```
>>> print sum([1, 2, 3, 4])
10
```

Juntando duas listas, formando pares de elementos:

```
>>> lista = zip(range(0, 5), range(5, 10))
>>> print lista
[(0, 5), (1, 6), (2, 7), (3, 8), (4, 9)]
```

Separando os elementos de uma lista de forma intercalada:

```
>>> lista = range(0, 10)
>>> intercaladas = lista[::2], lista[1::2]
>>> print intercaladas
([0, 2, 4, 6, 8], [1, 3, 5, 7, 9])
```

Transformando uma lista de strings em uma string CSV:

```
>>> lista = ["ola", "mundo", "aqui", "estamos"]
>>> csv_values = ','.join(lista)
>>> print csv_values
ola,mundo,aqui,estamos
```

Aplicando uma função (neste caso, anônima) a todos elementos de uma lista:

```
>>> lista = range(1, 11)
>>> print map(lambda x: x*-1, lista)
[-1, -2, -3, -4, -5, -6, -7, -8, -9, -10]
```

Filtrando os elementos de uma lista de acordo com um critério:

```
>>> def criterio(x): return x >= 0
>>> print range(-5, 5)
[-5, -4, -3, -2, -1, 0, 1, 2, 3, 4]
>>> print filter(criterio, range(-5, 5))
[0, 1, 2, 3, 4]
```

Retirando os elementos cujo valor é zero (ou melhor, cujo valor é avaliado como False):

```
>>> print filter(None, range(-2, 2))
[-2, -1, 1]
```


Capítulo 7

Strings

Além de números, Python também pode manipular strings (seqüências de caracteres), que podem ser expressas de diversas formas. Elas podem ser delimitadas por aspas simples ou duplas:

```
>>> 'spam eggs'
'spam eggs'
>>> 'doesn\'t'
"doesn't"
>>> "doesn't"
"doesn't"
>>> '"Yes," he said.'
'"Yes," he said.'
>>> "\"Yes,\" he said."
'"Yes," he said.'
>>> '"Isn\'t," she said.'
'"Isn\'t," she said.'
```

O interpretador exibe o resultado de operações com strings da mesma forma como elas são digitadas na entrada: dentro de aspas, e com aspas, caracteres acentuados e outros caracteres especiais representados por seqüências de escape com barras invertidas (como `'\t'`, `'\xc3\xa9'` etc.), para mostrar o valor preciso. A string é delimitada entre aspas simples, exceto quando ela contém uma aspa simples e nenhuma aspa dupla. O comando `print` produz uma saída mais legível para tais strings com caracteres especiais.

Strings que contém mais de uma linha podem ser construídas de diversas maneiras. Linhas de continuação podem ser usadas, com uma barra invertida colocada na última posição para indicar que a próxima linha física é a continuação de uma linha lógica:

```
oi = "Eis uma string longa contendo\n\
diversas linhas de texto assim como se faria em C.\n\
```

Finalmente, há o conceito de literais delimitadas por 3 aspas (simples ou duplas). Forma-se aqui o conceito de docstring, que é uma string eventualmente formada por mais de uma linha. As 3 aspas são necessárias porque sem elas há um erro na mudança de linha. Acompanhe

```
>>> print('ola

SyntaxError: EOL while scanning string literal (após dar o enter acima...)
>>> print('''ola
voces''')
ola
voces
>>>
```

A docstring deve descrever o que uma função do usuário faz. De acordo com o *Style Guide for Python Code* uma docstring deve começar com uma descrição em 1 linha (breve) seguida por uma linha em branco e depois detalhes adicionais, eventualmente exemplos e outras informações como o resultado retornado.

Strings podem ser indexadas; como em C, o primeiro caractere da string tem índice 0 (zero). Não existe um tipo específico para caracteres; um caractere é simplesmente uma string de tamanho 1. Assim como na linguagem Icon, substrings podem ser especificadas através da notação de slice (fatiamento ou intervalo): dois índices separados por dois pontos.

```
>>> palavra[4]
'a'
```

```
>>> palavra[0:2]
'Aj'
>>> palavra[2:4]
'ud'
```

Índices de fatias têm defaults úteis; a omissão do primeiro índice equivale a zero, a omissão do segundo índice equivale ao tamanho da string sendo fatiada.:

```
>>> palavra[:2]      # Os dois primeiros caracteres
'Aj'
>>> palavra[2:]     # Tudo menos os dois primeiros caracteres
'udaZ'
```

Diferentemente de C, strings em Python não podem ser alteradas. Tentar atribuir valor a uma posição (índice ou fatia) dentro de uma string resulta em erro:

```
>>> palavra[0] = 'x'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
>>> palavra[:1] = 'Splat'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

Entretanto, criar uma nova string combinando conteúdos é fácil e eficiente:

```
>>> 'x' + palavra[1:]
'xjudaZ'
>>> 'Splat' + palavra[5]
'SplatZ'
```

Eis uma invariante interessante das operações de fatiamento: $s[:i] + s[i:]$ é igual a s .

Intervalos fora de limites são tratados “graciosamente” (N.d.T: o termo original “gracefully” indica robustez no tratamento de erros): um índice maior que o comprimento é trocado pelo comprimento, um limite superior menor que o limite inferior produz uma string vazia.

```
>>> palavra[1:100]
'judaZ'
>>> palavra[10:]
''
>>> palavra[2:1]
''
```

Índices podem ser números negativos, para iniciar a contagem pela direita. Por exemplo:

```
>>> palavra[-1]     # O último caractere
'Z'
>>> palavra[-2]     # O penúltimo caractere
'a'
>>> palavra[-2:]    # Os dois últimos caracteres
'aZ'
>>> palavra[:-2]    # Tudo menos os dois últimos caracteres
'Ajud'
```

Observe que -0 é o mesmo que 0, logo neste caso não se conta a partir da direita!

```
>>> palavra[-0]
'A'
```

Intervalos fora dos limites da string são truncados, mas não tente isso com índices simples (que não sejam fatias):

```
>>> palavra[-100:]
'AjudaZ'
>>> palavra[-100]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
```

Uma maneira de lembrar como slices funcionam é pensar que os índices indicam posições entre caracteres, onde a borda esquerda do primeiro caractere é 0. Assim, a borda direita do último caractere de uma string de comprimento n tem índice n, por exemplo:

```
 0   1   2   3   4   5   6
+---+---+---+---+---+---+
| A | j | u | d | a | z |
+---+---+---+---+---+---+
-6  -5  -4  -3  -2  -1
```

A primeira fileira de números indica a posição dos índices 0...6 na string; a segunda fileira indica a posição dos respectivos índices negativos. Uma fatia de i a j consiste em todos os caracteres entre as bordas i e j, respectivamente.

Para índices positivos, o comprimento da fatia é a diferença entre os índices, se ambos estão dentro dos limites da string. Por exemplo, comprimento de palavra[1:3] é 2.

A função built-in (embutida) len() devolve o comprimento de uma string:

```
>>> s = 'anticonstitucionalissimamente'
>>> len(s)
29
```

Sequence Types — str, unicode, list, tuple, bytearray, buffer, xrange Strings, e as strings Unicode descritas na próxima seção, são exemplos de sequências e implementam as operações comuns associadas com esses objetos.

7.0.1 String Methods

Tanto strings comuns quanto Unicode oferecem um grande número de métodos para busca e transformações básicas. Eis alguns

.capitalize()

Transforma o primeiro caractere de uma string em maiúsculo. Ignora caracteres acentuados ou que não sejam letras.

```
>>> "curitiba".capitalize()
'Curitiba'
>>> "ágora".capitalize()
ágora
```

.center()

Centraliza a string usando espaços à esquerda e à direita, até atingir a largura.

```
>>> "paraná".center(20)
'      paraná      '
>>> "paraná".center(2)
'paraná'
```

.count()

Conta a quantidade de substrings dentro da string

```
>>> "abracadabra".count("ab")
2
>>> "abracadabra".count("a")
5
```

.find()

Retorna a primeira ocorrência do substring dentro da string. Caso nada seja encontrado, retorna -1.

```
>>> "abracadabra".count("ab")
2
>>> "abracadabra".count("a")
5
```

Avaliadores

São eles: `.isalpha()`, `.isdecimal()`, `.isdigit()`, `.islower()`, `.isnumeric()`, `.isspace()`, `.istitle()`, `.isupper()`. Retornam `True` caso a string seja composta exclusivamente de alfanuméricos, decimais, dígitos, minúsculos, numéricos, espaços, títulos (a primeira letra de cada palavra em maiúsculo) e maiúsculas. Todos ignoram as letras acentuadas.

`.join()`

Concatena todos os elementos dentro de iterável usando a string especificada como concatenador. A string pode ser vazia. Os elementos do iterável devem ser string.

```
>>> a=['agua', 'fogo', 'xabu']
>>> ''.join(a)
'aguafogoxabu'
>>> '\n'.join(a)
'agua\nfogo\nxabu'
>>> print('\n'.join(a))
agua
fogo
xabu
```

`.partition()`

Divide a string em uma tupla com 3 elementos: a parte anterior à primeira ocorrência do **separador**, o separador e a parte posterior ao separador

```
>>> "curitibaparanabrasil".partition('ra')
('curitibapa', 'ra', 'nabrasil')
>>> "curitibaparanabrasil".partition('z')
('curitibaparanabrasil', '', '')
```

`.replace()`

Substitui *n* ocorrências da substring **antiga** pela **nova**. Se *n* não é fornecido, ele substitui todas.

```
>>> "curitibaparanabrasil".replace("ra","agora")
'curitibapaagoranabagorasil'
>>> "curitibaparanabrasil".replace("ra","zz",1)
'curitibapazznabrasil'
```

`.split()`

Divide a string transformando-a em uma lista usando um separador *n* vezes. Se o separador não é informado ele usa um espaço em branco.

```
>>> "curitibaparanabrasil".split('a')
['curitib', 'p', 'r', 'n', 'br', 'sil']
>>> "curitiba parana brasil".split()
['curitiba', 'parana', 'brasil']
```

Muitos outros

Aqui estão só alguns, talvez os mais úteis e/ou principais. Consulte a documentação da versão de Python que você está usando. Estão todos lá.

7.1 Formatação

Existem basicamente 3 maneiras de formatar coisas em Python. A primeira é usando o caracter `%` que nos remete à formatação compatível com a linguagem C. Veja um exemplo

```
print(ct,("%6.3f   %6.4f   %10.5f   %10.5f"%(a,b,xm,(xb-xa))))
```


caracteres. Cada texto era tipicamente limitado a uma “code page” (uma tabela de códigos) que associava ordinais aos caracteres. Isso levou a muita confusão especialmente no âmbito da internacionalização de software (comumente escrito como i18n porque internationalization é 'i' + 18 letras + 'n'). Unicode resolve esses problemas ao definir uma única tabela de códigos para todos os conjuntos de caracteres.

Criar strings Unicode em Python é tão simples quanto criar strings normais:

```
>>> u'Hello World !'  
u'Hello World !'
```

O u antes das aspas indica a criação de uma string Unicode. Se você desejar incluir caracteres especiais na string, você pode fazê-lo através da codificação Unicode-Escape de Python. O exemplo a seguir mostra como:

```
>>> u'Hello\u0020World !'  
u'Hello World !'
```

O código de escape `\u0020` insere um caractere Unicode com valor ordinal 0x0020 (o espaço em branco) naquela posição.

Os outros caracteres são interpretados usando seus valores ordinais como valores ordinais em Unicode. Se você possui strings literais na codificação padrão Latin-1 que é usada na maioria dos países ocidentais, achará conveniente que os 256 caracteres inferiores do Unicode coincidem com os 256 caracteres do Latin-1.

Além dessas codificações padrão, Python oferece todo um conjunto de maneiras de se criar strings Unicode a partir de alguma codificação conhecida.

A função embutida `unicode()` dá acesso a todos os codecs Unicode registrados (COders e DECOders). Alguns dos codecs mais conhecidos são: Latin-1, ASCII, UTF-8, e UTF-16. Os dois últimos são codificações de tamanho variável para armazenar cada caractere Unicode em um ou mais bytes. (N.d.T: no Brasil, é muito útil o codec `cp1252`, variante estendida do Latin-1 usada na maioria das versões do MS Windows distribuídas no país, contendo caracteres comuns em textos, como aspas assimétricas “x” e ‘y’, travessão —, bullet • etc.).

Para converter uma string Unicode em uma string de 8-bits usando uma codificação específica, basta invocar o método `encode()` de objetos Unicode passando como parâmetro o nome da codificação destino. É preferível escrever nomes de codificação em letras minúsculas.

Se você tem um texto em uma codificação específica, e deseja produzir uma string Unicode a partir dele, pode usar a função `unicode()`, passando o nome da codificação de origem como segundo argumento.

7.1.3 Manipulação de bits

Embora seja algo meio fora de moda, eventualmente você vai precisar manusear bits em strings. Eis algumas funções

```
>>> bin(97)  
'0b1100001'  
>>> int('1100001',2)  
97  
>>> chr(97)  
'a'  
>>> ord('a')  
97  
>>> bin(ord('a'))  
'0b1100001'  
>>> chr(int('1100001',2))  
'a'  
>>> chr(int('01100001',2))  
'a'
```

Capítulo 8

Funções

Podemos criar uma função que escreve a série de Fibonacci até um limite arbitrário:

```
>>> def fib(n):    # escrever série de Fibonacci até n
...     """Exibe série de Fibonacci até n"""
...     a, b = 0, 1
...     while a < n:
...         print a,
...         a, b = b, a+b
...
>>> # Agora invocamos a função que acabamos de definir:
... fib(2000)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597
```

A palavra reservada `def` inicia a definição de uma função. Ela deve ser seguida do nome da função e da lista de parâmetros formais entre parênteses. Os comandos que formam o corpo da função começam na linha seguinte e devem ser indentados.

Opcionalmente, a primeira linha lógica do corpo da função pode ser uma string literal, cujo propósito é documentar a função. Se presente, essa string chama-se `docstring`. (Há mais informação sobre `docstrings` na seção `Strings de documentação`.) Existem ferramentas que utilizam `docstrings` para produzir automaticamente documentação online ou para imprimir, ou ainda permitir que o usuário navegue interativamente pelo código. É uma boa prática incluir sempre `docstrings` em suas funções, portanto, tente fazer disto um hábito.

A execução de uma função gera uma nova tabela de símbolos, usada para as variáveis locais da função. Mais precisamente, toda atribuição a variável dentro da função armazena o valor na tabela de símbolos local. Referências a variáveis são buscadas primeiramente na tabela local, então na tabela de símbolos global e finalmente na tabela de nomes embutidos (`built-in`). Portanto, não se pode atribuir diretamente um valor a uma variável global dentro de uma função (a menos que se utilize a declaração global antes), ainda que variáveis globais possam ser referenciadas livremente.

Os parâmetros reais (argumentos) de uma chamada de função são introduzidos na tabela de símbolos local da função no momento da invocação, portanto, argumentos são passados por valor (onde o valor é sempre uma referência para objeto, não o valor do objeto). [1] Quando uma função invoca outra, uma nova tabela de símbolos é criada para tal chamada.

Uma definição de função introduz o nome da função na tabela de símbolos atual. O valor associado ao nome da função tem um tipo que é reconhecido pelo interpretador como uma função definida pelo usuário. Esse valor pode ser atribuído a outros nomes que também podem ser usados como funções. Esse mecanismo serve para renomear funções:

```
>>> fib
<function fib at 10042ed0>
>>> f = fib
>>> f(100)
0 1 1 2 3 5 8 13 21 34 55 89
```

Conhecendo outras linguagens, você pode questionar que `fib` não é uma função, mas um procedimento, pois ela não devolve um valor. Na verdade, mesmo funções que não usam o comando `return` devolvem um valor, ainda que pouco interessante. Esse valor é chamado `None` (é um nome embutido). O interpretador interativo evita escrever `None` quando ele é o único resultado de uma expressão. Mas se quiser vê-lo pode usar o comando `print`:

```
>>> fib(0)
>>> print fib(0)
None
```

É fácil escrever uma função que devolve uma lista de números série de Fibonacci, ao invés de exibi-los:

```

>>> def fib2(n): # devolve a série de Fibonacci até n
...     """Devolve uma lista a com série de Fibonacci até n."""
...     resultado = []
...     a, b = 0, 1
...     while a < n:
...         resultado.append(a)    # veja mais adiante
...         a, b = b, a+b
...     return resultado
...
>>> f100 = fib2(100)    # executar
>>> f100                # exibir o resultado
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]

```

Este exemplo, como sempre, demonstra algumas características novas:

O comando `return` termina a função devolvendo um valor. Se não houver uma expressão após o `return`, o valor `None` é devolvido. Se a função chegar ao fim sem o uso explícito do `return`, então também será devolvido o valor `None`. O trecho `resultado.append(a)` invoca um método do objeto lista `resultado`. Um método é uma função que “pertence” a um objeto e é chamada através de `obj.nome_do_metodo` onde `obj` é um objeto qualquer (pode ser uma expressão), e `nome_do_metodo` é o nome de um método que foi definido pelo tipo do objeto. Tipos diferentes definem métodos diferentes. Métodos de diferentes tipos podem ter o mesmo nome sem ambiguidade. (É possível definir seus próprios tipos de objetos e métodos, utilizando classes, veja em *Classes*) O método `append()` mostrado no exemplo é definido para objetos do tipo lista; ele adiciona um novo elemento ao final da lista. Neste exemplo, ele equivale a `resultado = resultado + [a]`, só que mais eficiente.

8.1 Mais sobre definição de funções

É possível definir funções com um número variável de argumentos. Existem três formas, que podem ser combinadas.

8.1.1 Parâmetros com valores default

A mais útil das três é especificar um valor default para um ou mais parâmetros formais. Isso cria uma função que pode ser invocada com um número menor de argumentos do que ela pode receber. Por exemplo:

```

def confirmar(pergunta, tentativas=4, reclamacao='Sim ou não, por favor!'):
    while True:
        ok = raw_input(pergunta).lower()
        if ok in ('s', 'si', 'sim'):
            return True
        if ok in ('n', 'no', 'não', 'nananinão'):
            return False
        tentativas = tentativas - 1
        if tentativas == 0:
            raise IOError('usuario nao quer cooperar')
    print reclamacao

```

Essa função pode ser invocada de várias formas:

- fornecendo apenas o argumento obrigatório: `confirmar('Deseja mesmo encerrar?')`
- fornecendo um dos argumentos opcionais: `confirmar('Sobrescrever o arquivo?', 2)` ou
- fornecendo todos os argumentos: `confirmar('Sobrescrever o arquivo?', 2, 'Escolha apenas s ou n')`

Este exemplo também introduz o operador `in`, que verifica se uma sequência contém ou não um determinado valor.

Os valores default são avaliados no momento a definição da função, e no escopo em que a função foi definida, portanto:

```

i = 5

def f(arg=i):
    print arg

i = 6
f()
irá exibir 5.

```

Aviso importante: Valores default são avaliados apenas uma vez. Isso faz diferença quando o valor default é um objeto mutável como uma lista ou dicionário (N.d.T. dicionários são como arrays associativos ou HashMaps em outras linguagens; ver Dicionários).

Por exemplo, a função a seguir acumula os argumentos passados em chamadas subsequentes:

```
def f(a, L=[]):
    L.append(a)
    return L

print f(1)
print f(2)
print f(3)
```

Esse código vai exibir:

```
[1]
[1, 2]
[1, 2, 3]
```

Se você não quiser que o valor default seja compartilhado entre chamadas subsequentes, pode reescrever a função assim:

```
def f(a, L=None):
    if L is None:
        L = []
    L.append(a)
    return L
\subsection{Argumentos nomeados}
Funções também podem ser chamadas passando keyword arguments (argumentos nomeados) no formato
\begin{verbatim}
def parrot(voltage, state='a stiff', action='vroom', type='Norwegian Blue'):
    print "-- This parrot wouldn't", action,
    print "if you put", voltage, "volts through it."
    print "-- Lovely plumage, the", type
    print "-- It's", state, "!"
```

aceita um argumento obrigatório (voltage) e três argumentos opcionais (state, action, e type). Esta função pode ser invocada de todas estas formas:

```
parrot(1000) # 1 arg. posicional
parrot(voltage=1000) # 1 arg. nomeado
parrot(voltage=1000000, action='VOOOOOM') # 2 arg. nomeados
parrot(action='VOOOOOM', voltage=1000000) # 2 arg. nomeados
parrot('a million', 'bereft of life', 'jump') # 3 arg. posicionais
# 1 arg. posicional e 1 arg. nomeado
parrot('a thousand', state='pushing up the daisies')
```

mas todas as invocações a seguir seriam inválidas:

```
parrot() # argumento obrigatório faltando
parrot(voltage=5.0, 'dead') # argumento posicional depois do nomeado
parrot(110, voltage=220) # valor duplicado para o mesmo argument
parrot(actor='John Cleese') # argumento nomeado desconhecido
```

Em uma invocação, argumentos nomeados devem vir depois dos argumentos posicionais. Todos os argumentos nomeados passados devem casar com os parâmetros formais definidos pela função (ex. actor não é um argumento nomeado válido para a função parrot), mas sua ordem é irrelevante. Isto também inclui argumentos obrigatórios (ex.: parrot(voltage=1000) funciona). Nenhum parâmetro pode receber mais de um valor. Eis um exemplo que não funciona devido a esta restrição:

```
>>> def function(a):
...     pass
...
>>> function(0, a=0)
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: function() got multiple values for keyword argument 'a'
```

Quando o último parâmetro formal usar a sintaxe `**nome`, ele receberá um dicionário (ver Dicionários ou Mapping Types — dict [online]) com todos os parâmetros nomeados passados para a função, exceto aqueles que corresponderam a parâmetros formais definidos antes. Isto pode ser combinado com o parâmetro formal `*nome` (descrito na próxima subseção) que recebe uma tupla (N.d.T. uma sequência de itens, semelhante a uma lista imutável; ver Tuplas e sequências) contendo todos argumentos posicionais que não correspondem à lista da parâmetros formais. (`*nome` deve ser declarado antes de `**nome`.) Por exemplo, se definimos uma função como esta:

```
def cheeseshop(kind, *arguments, **keywords):
    print "-- Do you have any", kind, "?"
    print "-- I'm sorry, we're all out of", kind
    for arg in arguments:
        print arg
    print "-" * 40
    keys = sorted(keywords.keys())
    for kw in keys:
        print kw, ":", keywords[kw]
```

Ela pode ser invocada assim:

```
cheeseshop("Limburger", "It's very runny, sir.",
           "It's really very, VERY runny, sir.",
           shopkeeper='Michael Palin',
           client="John Cleese",
           sketch="Cheese Shop Sketch")
```

e, naturalmente, produziria:

```
-- Do you have any Limburger ?
-- I'm sorry, we're all out of Limburger
It's very runny, sir.
It's really very, VERY runny, sir.
-----
client : John Cleese
shopkeeper : Michael Palin
sketch : Cheese Shop Sketch
```

Note que criamos uma lista de chaves `keys` ordenando o resultado do método `keys()` do dicionário `keywords` antes de exibir seu conteúdo; se isso não fosse feito, os argumentos seriam exibidos em uma ordem não especificada.

8.1.2 Listas arbitrárias de argumentos

Finalmente, a opção menos usada possibilita que função seja invocada com um número arbitrário de argumentos. Esses argumentos serão empacotados em uma tupla (ver Tuplas e sequências). Antes dos argumentos em número variável, zero ou mais argumentos normais podem estar presentes.

```
def escrever_multiplos_itens(arquivo, separador, *args):
    arquivo.write(separador.join(args))
```

8.1.3 Desempacotando listas de argumentos

A situação inversa ocorre quando os argumentos já estão numa lista ou tupla mas ela precisa ser explodida para invocarmos uma função que requer argumentos posicionais separados. Por exemplo, a função `range()` espera argumentos separados, `start` e `stop`. Se os valores já estiverem juntos em uma lista ou tupla, escreva a chamada de função com o operador `*` para desempacotá-los da sequência:

```
>>> range(3, 6)      # chamada normal com argumentos separados
[3, 4, 5]
>>> args = [3, 6]
>>> range(*args)     # chamada com argumentos desempacotados de uma lista
[3, 4, 5]
```

Da mesma forma, dicionários podem produzir argumentos nomeados com o operador `**`:

```
>>> def parrot(voltage, state='a stiff', action='voom'):
...     print "-- This parrot wouldn't", action,
...     print "if you put", voltage, "volts through it.",
...     print "E's", state, "!"
```

```

...
>>> d = {"voltage": "four million", "state": "bleedin' demised", "action": "VOOM"}
>>> parrot(**d)

```

– This parrot wouldn't VOOM if you put four million volts through it. E's bleedin' demised !

8.1.4 Passagem por valor ou por referência

Em muitas linguagens há uma distinção formal e importante entre parâmetros passados por valor e por referência. Lá nas outras linguagens, a passagem por valor gera uma cópia do parâmetro e passa esta cópia para a função. Este mecanismo preserva a variável original com seu valor original que permanece a despeito de toda e qualquer modificação que a função faça nela.

Por outro lado (ainda naquelas linguagens...) a passagem por referência implica em que o que é passado para a função é o endereço de memória da variável. Daqui, que qualquer modificação que a função fizer sobre ela, será feita sobre a única instância da variável, e portanto qualquer modificação que a função fizer sobre ela feita estará.

Aqui em Python esta discussão não se coloca (não há como comandar esta passagem, como por exemplo colocando **var** antes do nome em pascal, ou colocando um **&** antes do nome em C/C++). Aqui em Python, a distinção se dá entre parâmetros mutáveis e imutáveis. Quando a função recebe um parametro imutável e o modifica, claramente a mudança é feita sobre uma cópia do parâmetro. Tudo funciona então como se a passagem tivesse sido por valor.

Já, se o parâmetro passado for mutável, e a função o modificar, a variável estará modificada e neste caso tudo funciona como se a passagem fosse por referência. Acompanhe no exemplo

```

import numpy as np
def fu(x):
    x[1,1]=22
    return
a=np.array([[1,2,3],[4,5,6],[7,8,9]])
fu(a)
print(a)
[[ 1  2  3]
 [ 4 22  6]
 [ 7  8  9]]

```

8.1.5 Construções lambda

Atendendo a pedidos, algumas características encontradas em linguagens de programação funcionais como Lisp foram adicionadas a Python. Com a palavra reservada lambda, pequenas funções anônimas podem ser criadas. Eis uma função que devolve a soma de seus dois argumentos:

```
lambda a, b: a+b.
```

Construções lambda podem ser empregadas em qualquer lugar que exigiria uma função. Sintaticamente, estão restritas a uma única expressão. Semanticamente, são apenas açúcar sintático para a definição de funções normais. Assim como definições de funções aninhadas, construções lambda podem referenciar variáveis do escopo onde são definidas (N.d.T isso significa que Python implementa closures, recurso encontrado em Lisp, JavaScript, Ruby etc.):

```

>>> def fazer_incrementador(n):
...     return lambda x: x + n
...
>>> f = fazer_incrementador(42)
>>> f(0)
42
>>> f(1)
43

```

8.1.6 Strings de documentação

A comunidade Python está convencionando o conteúdo e o formato de strings de documentação (docstrings).

A primeira linha deve ser um resumo curto e conciso do propósito do objeto. Por brevidade, não deve explicitamente se referir ao nome ou tipo do objeto, uma vez que estas informações estão disponíveis por outros meios (exceto se o nome da função for o próprio verbo que descreve a finalidade da função). Essa linha deve começar com letra maiúscula e terminar com ponto.

Se existem mais linhas na string de documentação, a segunda linha deve estar em branco, separando visualmente o resumo do resto da descrição. As linhas seguintes devem conter um ou mais parágrafos descrevendo as convenções de chamada ao objeto, seus efeitos colaterais, etc.

O parser do Python não remove a indentação de comentários multi-linha. Portanto, ferramentas que processem strings de documentação precisam lidar com isso, quando desejável. Existe uma convenção para isso. A primeira linha não vazia após a linha de sumário determina a indentação para o resto da string de documentação. (Não podemos usar a primeira linha para isso porque ela em geral está adjacente às aspas que iniciam a string, portanto sua indentação real não fica aparente.) Espaços em branco ou tabs “equivalentes” a esta indentação são então removidos do início das demais linhas da string. Linhas indentação menor não devem ocorrer, mas se ocorrerem, todos os espaços à sua esquerda são removidos. Para determinar a indentação, normalmente considera-se que um caractere tab equivale a 8 espaços.

Eis um exemplo de uma docstring multi-linha:

```
>>> def minha_funcao():
...     """Não faz nada, mas é documentada.
...
...     Realmente ela não faz nada.
...     """
...     pass
...
>>> print minha_funcao.__doc__
Não faz nada, mas é documentada.
```

Realmente ela não faz nada.

Capítulo 9

Objetos em Python

Historicamente o paradigma de objetos nasceu como uma das reações da comunidade de software ao problema cada vez mais importante da complexidade crescente do software. É só você olhar para os lados (bancos, companhias aéreas, governos, grandes corporações, hospitais, um carro moderno, etc etc) para ver que a humanidade literalmente lança seu futuro nas mãos do software e seja o que Deus quiser.

A idéia do objeto é encapsular (esconder, impedir que alguém indeterminado faça alterações, coisas desse estilo) dados e procedimentos de maneira a garantir a integridade de uns e de outros. Dito de outra maneira, o paradigma de objetos interpõe um intermediário (o objeto) entre os ativos tecnológicos (dados e procedimentos) e seus usuários (programas e programadores que os manipulam).

Isto é necessário já que é uma evidência empírica que boa parte dos bugs (erros) de programa derivam de efeitos colaterais de manipulação inadvertida de áreas comuns de dados dentro dos programas. Sobretudo, se não são os maiores erros em número, certamente são os mais difíceis de corrigir por (aparentemente) muito pouco terem a ver com as ações que os causaram. Em outras palavras quando a relação causa \Leftrightarrow efeito de um erro é tênue ou escondida, sua correção demora e custa muito mais.

A explicação parece complexa e o é. Razão pela qual – na minha opinião – a metodologia de objetos se aplica maravilhosamente a sistemas grandes e/ou complexos, mas para aprender a programar, ou mesmo para aplicações simples é um exagero seu uso. É como matar um mosquito usando uma bala de canhão.

Mas aqui, de novo, a superioridade de Python: ele pode ser aprendido e usado o tempo todo como uma linguagem imperativa: como se viu este é o melhor jeito de aprender a programar. Quando (e se) lá na frente, você sentir falta das qualidades inerentes ao paradigma orientado ao objeto, *voilà*, não é que Python é 100% orientado ao objeto?

Dito isto, um objeto em Python é uma entidade que tem dados e procedimentos associados, e devidamente encobertos (encapsulados). Um objeto é definido em python pelo uso da cláusula *class*. Veja como:

```
class carro:
```

Note que a definição termina por **:** o que caracteriza que ela continua. A próxima coisa é definir um objeto python de nome `__init__`. Veja

```
    def __init__(self):
```

Este objeto é o *construtor* e ele é automaticamente chamado pelo python toda vez que um objeto é instanciado (definido e usado pela primeira vez). A variável *self* é o objeto propriamente dito. Note que o construtor é uma função, pelo que também termina por **:**

Nesta função construtora definem-se os valores iniciais do objeto que passarão a fazer parte dele assim que o objeto for criado (instanciado). Acompanhe no exemplo completo

```
class carro:
    def __init__(self):
        self.kilo = 0
        self.marca = 'peugeot'
        self.portas = 5
```

Aqui definiu-se um objeto *carro* e disse-se que a kilometragem do carro é 0, a marca é peugeot e a quantidade de portas é 5.

Depois dessa definição, pode-se criar quantos objetos se quiserem, como em

```
carro1 = carro()
carro2 = carro()
carro2.portas = 3
```

Neste caso, há 2 carros: *carro1* e *carro2*. Em particular, o *carro2* tem um número diferente de portas, a saber, 3. Acompanhe

```
print(carro1.portas)
print(carro2.portas)
```

O python responderá 5 para a primeira impressão e 3 para a segunda.

9.1 Um exemplo de objeto: um formigueiro

A seguir uma prova de como é poderoso o conceito de objeto. Eis uma possível simulação de um formigueiro, que pode ser descrito como

- Uma coletividade de inúmeras (50, 100, 200 ? quantas ?) formigas
- As formigas são similares, isto é, todas têm o mesmo comportamento,
- Mas cada uma é um indivíduo separado das demais, e nesta implementação, uma não interfere na outra.

Se não se usasse a orientação a objeto, haveria que programar a coisa na unha, controlando para cada formiga seus dados, e lutando bravamente para que os dados de cada formiga não fossem (inadvertidamente) alterados por outra formiga. O software seria grande, complexo, sujeito a largos processos de depuração e correção.

Ao usar objetos, veja-se as vantagens adquiridas.

- Programa-se apenas 1 formiga. Não importa quantas formigas existam. Este enfoque permite que o comportamento dessa formiga seja tão complexo e rebuscado quanto se queira. Afinal é só uma formiga.
- Cada formiga enxerga seus dados e pode manipulá-los sem medo. Não há como uma formiga alterar dados de outra: lembre-se que no programa só há uma formiga.
- A criação de várias formigas é tarefa simplória: Definido o objeto, muitas formigas são criadas em um bloco for, que gera instâncias diferentes da mesma formiga. Agora, passar de 10 a 1000 formigas é apenas mudar o valor final de um laço for.
- O programa fica menor. Seus erros de implementação são sensivelmente menores e portanto sua implementação mais rápida.
- Os bugs misteriosos não aparecem.

```
# exemplo de objetos (formigas em um gramado)
import turtle, random
class formiga:
    def __init__(self,i): #aqui define-se o comportamento inicial da formiga
        # um parâmetro passado é um número i
        self.t=turtle.Pen() #cria-se uma turtle de nome t no objeto e ela tem o método pen
        tcor=["violet","red","yellow","green","blue","brown","orange","pink","black",
            "purple","olive","fuchsia","blueviolet","plum",
            "coral","cornsilk","chartreuse","cyan","aqua","crimson"] #as diversas cores
        # ind=random.randint(0,7)
        self.t.color=tcor[i%len(tcor)] #a cor é atribuída de maneira circular
        self.t.ondex=200-random.randint(0,399) #a formiga começa em x= -200..200 randomicamente
        self.t.ondey=200-random.randint(0,399) # idem para y
        self.t.direcao=random.randint(0,359) # ganha uma direção randômica 0..360
        self.t.potencia=random.randint(1,11) # ganha uma potência randômica 1..11
        self.t.penup() # levanta a caneta
        self.t.goto(self.t.ondex,self.t.ondey) # vai para onde ela está
        self.t.pendown() # abaixa a caneta

    def anda(self): # como a formiga anda
        self.t.penup() # levanta a caneta
        self.t.goto(self.t.ondex,self.t.ondey) # vai para onde ela está
        self.t.pensize(self.t.potencia) # associa a grossura do traço à potência
        self.t.pendown() # abaixa a caneta
        self.t.pencolor(self.t.color) # associa a cor da caneta
        dist=random.randint(25,70) # dist= randomica 25..70
        ndir=random.randint(0,359) # ndir= direção randômica 0..359
        self.t.right(ndir) # manda girar à direita ndir
        self.t.fd(dist) # manda andar dist
        x=self.t.pos() # descobre onde parou (x,y)
```

```
self.t.ondex=x[0]                # guarda onde parou (x)
self.t.ondey=x[1]                # guarda onde parou (y)

x=[0]*60 # criam-se 60 formigas
for i in range(0,len(x)):
    x[i]=formiga(i) # cada formiga ganha a cor i
    x[i].anda()    # anda
for j in range(len(x)): realizam-se 14400 (60*60*4) movimentos
    for i in range(len(x)): # cada formiga (são 60) realiza 4 movimentos a cada ciclo
        x[i].anda()
    for i in range(len(x)):
        x[i].anda()
    for i in range(len(x)):
        x[i].anda()
    for i in range(len(x)):
        x[i].anda()
```

9.1.1 Herança

Em uma relação de herança, existem no mínimo 2 classes envolvidas: A mais genérica chamada superclasse ou classe pai. A outra é a mais específica, chamada de subclasse, classe filha ou ainda derivada. É comum em uma classe derivada sobrecarregar os métodos da classe pai.

Capítulo 10

Arquivos

A função `open()` devolve um objeto arquivo, e é frequentemente usada com dois argumentos: `open(nome_do_arquivo, modo)`.

```
>>> f = open('/tmp/workfile', 'w')
>>> print f
<open file '/tmp/workfile', mode 'w' at 80a0960>
```

O primeiro argumento é uma string contendo o nome do arquivo. O segundo argumento é outra string contendo alguns caracteres que descrevem o modo como o arquivo será usado. O parâmetro `mode` pode ser `'r'` quando o arquivo será apenas lido, `'w'` para escrever (se o arquivo já existir seu conteúdo prévio será apagado), e `'a'` para abrir o arquivo para adição; qualquer escrita será adicionada ao final do arquivo. A opção `'r+'` abre o arquivo tanto para leitura como para escrita. O parâmetro `mode` é opcional, em caso de omissão será assumido `'r'`.

No Windows, `'b'` adicionado a string de modo indica que o arquivo será aberto em modo binário. Sendo assim, existem os modos compostos: `'rb'`, `'wb'`, e `'r+b'`. O Windows faz distinção entre arquivos texto e binários: os caracteres terminadores de linha em arquivos texto são alterados ao ler e escrever. Essa mudança automática é útil em arquivos de texto ASCII, mas corrompe arquivos binários como `.JPEG` ou `.EXE`. Seja cuidadoso e use sempre o modo binário ao manipular tais arquivos. No Unix, não faz diferença colocar um `'b'` no modo, então você pode usar isto sempre que quiser lidar com arquivos binários de forma independente da plataforma.

N.d.T. Para ler arquivos de texto contendo acentuação e outros caracteres não-ASCII, a melhor prática desde o Python 2.6 é usar a função `io.open()`, do módulo `io`, em vez da função embutida `open()`. O motivo é que `io.open()` permite especificar a codificação logo ao abrir um arquivo em modo texto para leitura ou escrita. Desta forma, a leitura do arquivo texto sempre devolverá objetos `unicode`, independente da codificação interna do arquivo no disco. E ao escrever em um arquivo aberto via `io.open()`, basta enviar strings `unicode`, pois a conversão para o encoding do arquivo será feita automaticamente. Note que ao usar `io.open()` sempre faz diferença especificar se o arquivo é binário ou texto, em todos os sistemas operacionais: o modo texto é o default, mas se quiser ser explícito coloque a letra `t` no parâmetro `mode` (ex. `rt`, `wt` etc.); use a letra `b` (ex. `rb`, `wb` etc.) para especificar modo binário. Somente em modo texto os métodos de gravação e leitura aceitam e devolvem strings `unicode`. Em modo binário, o método `write` aceita strings de bytes, e os métodos de leitura devolvem strings de bytes também.

10.1 Métodos de objetos arquivo

Para simplificar, o resto dos exemplos nesta seção assumem que um objeto arquivo chamado `f` já foi criado.

Para ler o conteúdo de um arquivo, invoque `f.read(size)`, que lê um punhado de dados devolvendo-os como uma string de bytes `str`. O argumento numérico `size` é opcional. Quando `size` é omitido ou negativo, todo o conteúdo do arquivo é lido e devolvido; se o arquivo é duas vezes maior que memória da máquina, o problema é seu. Caso contrário, no máximo `size` bytes serão lidos e devolvidos. Se o fim do arquivo for atingido, `f.read()` devolve uma string vazia `()`.

```
>>> f.read()
'Texto completo do arquivo.\n'
>>> f.read()
''
```

O método `f.readline()` lê uma única linha do arquivo; o caractere de quebra de linha (`'\n'`) é mantido ao final da string, só não ocorrendo na última linha do arquivo, se ela não termina com uma quebra de linha. Isso elimina a ambiguidade do valor devolvido; se `f.readline()` devolver uma string vazia, então é certo que o arquivo acabou. Linhas em branco são representadas por um `'\n'` – uma string contendo apenas o terminador de linha.

```
>>> f.readline()
'Primeira linha do arquivo.\n'
```

```
>>> f.readline()
'Segunda linha do arquivo.\n'
>>> f.readline()
''
```

O método `f.readlines()` devolve uma lista contendo todas as linhas do arquivo. Se for fornecido o parâmetro opcional `sizehint`, será lida a quantidade especificada de bytes e mais o suficiente para completar uma linha. Frequentemente, isso é usado para ler arquivos muito grandes por linhas, sem ter que ler todo o arquivo para a memória de uma só vez. Apenas linhas completas serão devolvidas.

```
>>> f.readlines()
['Primeira linha do arquivo.\n', 'Segunda linha do arquivo.\n']
```

Uma maneira alternativa de ler linhas do arquivo é iterar diretamente pelo objeto arquivo. É eficiente, rápido e resulta em código mais simples:

```
>>> for line in f:
    print line,
```

gerará o seguinte resultado

```
Primeira linha do arquivo.
Segunda linha do arquivo.
```

Essa alternativa é mais simples, mas não oferece tanto controle. Como as duas maneiras gerenciam o buffer do arquivo de modo diferente, elas não devem ser misturadas.

O método `f.write(string)` escreve o conteúdo da string de bytes para o arquivo, devolvendo `None`.

```
>>> f.write('Isto é um teste.\n')
```

N.d.T. Neste exemplo, a quantidade de bytes que será escrita no arquivo vai depender do encoding usado no console do Python. Por exemplo, no encoding UTF-8, a string acima tem 18 bytes, incluindo a quebra de linha, porque são necessários dois bytes para representar o caractere 'é'. Mas no encoding CP1252 (comum em Windows no Brasil), a mesma string tem 17 bytes. O método `f.write` apenas escreve bytes; o que eles representam você decide.

Ao escrever algo que não seja uma string de bytes, é necessário converter antes:

```
>>> valor = ('a resposta', 42)
>>> s = str(valor)
>>> f.write(s)
```

N.d.T. Em particular, se você abriu um arquivo `f` com a função embutida `open()`, e deseja escrever uma string Unicode `x` usando `f.write`, deverá usar o método `unicode.encode()` explicitamente para converter `x` do tipo `unicode` para uma string de bytes `str`, deste modo: `f.write(x.encode('utf-8'))`. Por outro lado, se abriu um arquivo `f2` com `io.open()`, pode usar `f2.write(x)` diretamente, pois a conversão de `x` – de `unicode` para o encoding do arquivo – será feita automaticamente.

O método `f.tell()` devolve um inteiro `long` que indica a posição atual de leitura ou escrita no arquivo, medida em bytes desde o início do arquivo. Para mudar a posição utilize `f.seek(offset, de_onde)`. A nova posição é computada pela soma do deslocamento `offset` a um ponto de referência especificado pelo argumento `de_onde`. Se o valor de `de_onde` é 0, a referência é o início do arquivo, 1 refere-se à posição atual, e 2 refere-se ao fim do arquivo. Este argumento pode ser omitido; o valor default é 0.

```
>>> f = open('/tmp/workfile', 'r+')
>>> f.write('0123456789abcdef')
>>> f.seek(5)      # Vai para o sexto byte do arquivo
>>> f.read(1)
'5'
>>> f.seek(-3, 2) # Vai para terceiro byte antes do fim
>>> f.read(1)
'd'
```

Quando acabar de utilizar o arquivo, invoque `f.close()` para fechá-lo e liberar recursos do sistema (buffers, descritores de arquivo etc.). Qualquer tentativa de acesso ao arquivo depois dele ter sido fechado resultará em falha.

```
>>> f.close()
>>> f.read()
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
ValueError: I/O operation on closed file
```

É uma boa prática usar o comando `with` ao lidar com objetos arquivo. Isto tem a vantagem de garantir que o arquivo seja fechado quando a execução sair do bloco dentro do `with`, mesmo que uma exceção tenha sido levantada. É também muito mais sucinto do que escrever os blocos `try-finally` necessários para garantir que isso aconteça.

```
>>> with open('/tmp/workfile', 'r') as f:
...     read_data = f.read()
>>> f.closed
True
```

Objetos arquivo têm métodos adicionais, como `isatty()` e `truncate()` que são usados com menos frequência; consulte a Referência da Biblioteca Python para mais informações.

10.2 Um exemplo de criação de arquivo

```
import sympy as sp
import numpy as np
import random
arq=open("c:/apl2/exe078.txt",'w')
for i in range(300):
    x = np.random.randint(1,7,(2,2))
    print(x.reshape(4))
    arq.write(str(x.reshape(4))+'\n')
    y=sp.Matrix(x)
    q,lam=y.diagonalize()
    print(lam[0,0], ' ', lam[1,1])
    arq.write(str(lam[0,0])+'\n')
    arq.write(str(lam[1,1])+'\n')
arq.close()
```

10.3 O módulo pickle

Strings podem ser facilmente escritas e lidas de um arquivo. Números exigem um pouco mais de esforço, uma vez que o método `read()` só devolve strings, obrigando o uso de uma função como `int()` para produzir o número 123 a partir da string '123'. Entretanto, quando estruturas de dados mais complexas (listas, dicionários, instâncias de classe, etc) estão envolvidas, o processo se torna bem mais complicado.

Para não obrigar os usuários a escrever e depurar constantemente código para salvar estruturas de dados, Python oferece o módulo padrão `pickle`. Este é um módulo que permite converter praticamente qualquer objeto Python (até mesmo certas formas de código!) para uma string de bytes. Este processo é denominado `pickling` (N.d.T. literalmente, “colocar em conserva”, como picles de pepinos em conserva). E `unpickling` é o processo reverso: reconstruir o objeto a partir de sua representação como string de bytes. Enquanto estiver representado como uma string, o objeto pode ser facilmente armazenado em um arquivo ou banco de dados, ou transferido pela rede para uma outra máquina.

Se você possui um objeto qualquer `x`, e um objeto arquivo `f` que foi aberto para escrita, a maneira mais simples de utilizar este módulo é:

```
pickle.dump(x, f)
```

Para reconstruir o objeto `x`, sendo que `f` agora é um arquivo aberto para leitura:

```
x = pickle.load(f)
```

(Existem outras variações desse processo, úteis quando se precisa aplicar sobre muitos objetos ou o destino da representação string não é um arquivo; consulte a documentação do módulo `pickle` na Referência da Biblioteca Python.)

O módulo `pickle` é a forma padrão de fazer objetos Python que possam ser compartilhados entre diferentes programas Python, ou pelo mesmo programa em diferentes sessões de execução; o termo técnico para isso é objeto persistente. Justamente porque o módulo `pickle` é amplamente utilizado, vários autores que escrevem extensões para Python tomam o cuidado de garantir que novos tipos de dados, como matrizes numéricas, sejam compatíveis com esse processo.

Capítulo 11

Pacotes em Python

11.1 Conceito

Esta é uma discussão velha como a história do mundo: generalidade X especificidade. No caso de uma linguagem de programação, esta questão ressurgue forte: afinal, ao entregar uma ferramenta capaz de programar dispositivos “inteligentes”, quer-se uma linguagem que atenda a todos os dispositivos (generalidade), mas que converse de perto e com intimidade com qualquer dispositivo (especificidade). Este dilema não é específico de Python. Em Java, por exemplo, esta equação foi resolvida na origem, pela introdução de uma máquina virtual entre o programa e o dispositivo. Note-se que esta questão só surge nas linguagens mais modernas. Falar disto em Lisp ou em Fortran é completamente fora de propósito: na época em que estas linguagens surgiram só se programava um mainframe e olhe lá (ainda que fosse um mainframe de 8K de memória principal, caso do IBM1130 com Fortran).

Voltando ao Python, aqui a equação foi resolvida mantendo o núcleo central do Python pequeno e ágil (generalidade) e dispondo de toda a funcionalidade específica de alguma arquitetura ou aplicação separada deste núcleo, embora facilmente integrável a ele. Estamos falando dos pacotes Python.

Os pacotes python moram num local chamado PYPI (Python Package Index, disponível em <https://pypi.org/>). Eis os números deste índice em 16/04/2024: 530.560 projetos, 5.584.105 lançamentos, 10.805.292 arquivos e 806.048 usuários.

11.2 Instalação

Cada ambiente de Python terá sua especificidade para instalar pacotes. Vai se ver aqui a instalação sob windows, em máquinas nas quais o operador tenha ou possa vir a ter a credencial de administrador. Para tanto acompanhe o passo a passo:

1. Descubra em que diretório o Python foi instalado. Por causa desta necessidade aqui apontada é que se sugere, na instalação do Python, definir um diretório próximo à raiz do disco e com mneumônico fácil. Na minha máquina, por exemplo, o Python está instalado em `c:\pyt312`.
2. Neste diretório, desça um nível em direção ao sub-diretório `scripts`. O comando em dos/windows é `cd scripts<enter>`.
3. Uma vez lá, emita o comando de instalação do pacote que é `pip install <pacote>`, onde `<pacote>` deve ser substituído pelo nome do pacote que se quer instalar. Por exemplo `pip install numpy`.
4. Deve-se aguardar pela instalação do pacote e de todos os pré-requisitos que este pacote apresenta. Geralmente vários pacotes são instalados automaticamente, antes de instalar o que você pediu.
5. Para verificar se deu certo, quando este processo acabar, entre em uma sessão python pelo método usual e lá dentro emita o comando `import <pacote>`, como por exemplo `import numpy`. Se tal comando retornar apenas o prompt `>>>` isto significa que a instalação e a importação do pacote deram certo.

11.3 uma amostra

Uma pequena amostra dos mais de 100.000 pacotes existentes no ambiente Python

11.4 LISTA DE PACOTES PYTHON

PACOTE	FINALIDADE
--------	------------

TENSORFLOW	Implementa redes neurais em python. Autoria: google
PYEASYGA	Implementa algoritmos de algoritmos genéticos
DEAP	Implementa algoritmos de computação evolutiva, incluindo GA
BOKEH	Apresentação de dados: tabelas, desenhos e inúmeras formas de apresentação de dados
MATPLOTLIB	Apresentação de dados
PYGAME	Toda a infraestrutura para escrever games em python
TKINTER	Interface TK para escrever aplicativos gráficos em python
NUMPY	Computação numérica em python
TURTLE	Simulação da linguagem LOGO
RANDOM	Todas as tratativas para manuseio de números aleatórios
ASTROPY	manuseia arquivos digitais FITS
DJANGO	Framework web
FLASK	Framework web
HUG	Framework web
KERAS	Machine learning, mais fácil que o tensor flow
SCIKIT-LEARN	Machine learning
THEANO	Machine learning
OpenCV	Visão computacional
ORANGE3	Framework para data mining
SCRAPY	Um crawler (spider fuçadeira) em python
PYODBC	ODBC em python (para acessar bancos de dados)
SPHINX	Documentador em python (originalmente desenvolvido para documentar o python, mas pode ser usado em outros ambientes)
REQUESTS	Requisições HTTP
COLLECTIONS	Mantém histórico de processamentos em contêdores. Elabora diversos processamentos auxiliares em listas.
HEAPQ	recupera os n maiores ou menores em uma coleção usando um <i>heap</i>
RE	Processamento extensivo de expressões regulares, por exemplo pesquisando e substituindo strings.
UNICODEDATA	Normalização e compatibilização de strings UTF-8, UTF-16, ASCII, Latin1, entre muitos outros.
TEXTWRAP	reformatar texto em diversas modalidades.
DECIMAL	Implementa a <i>general decimal arithmetic specification</i> da IBM.
CMATH	Aritmética e álgebra complexa.
RANDOM	Funções para números aleatórios.
DATETIME	conversões e aritmética envolvendo diferentes unidades de tempo.
ITERTOOLS	ferramentas adicionais de iteração. Por exemplo, ele gera todas as combinações ou permutações de um contêdor.
GZIP e BZ2	Comprime e descomprime conteúdos usando GZIP ou BZ2.
FNMATCH	processa nomes de arquivos e diretórios.
OS	interfaceia com o sistema operacional.
PYSERIAL	Manusear a interface serial do computador.
PICKLE	Serializar objetos.
CSV	Manusear arquivos tipo CSV.
JSON	Manusear dados JSON.
XML	Manusear objetos XML.
BASE64	Converter de e para Base64.
STRUCT	Manusear objetos binários complexos, registros variáveis aninhados, etc.
PANDAS	Análise de dados envolvendo estatísticas, séries temporais e técnicas relacionadas.
FUNCTOOLS	Ferramentas adicionais para definição de funções. (como wrapper=encapsular uma função qualquer com processamento adicional, por exemplo um timmer).
QUEUE	manipulação de filas e listas.
WEAKREF	Indireções com referências fracas (compatibilidade com gabrage collection).
AST	Abstract Syntax Tree, ou criar AST a partir de código fonte Python, permitindo análise e manipulação.
DIS	disassembly de código python.
SYS	Manipulação de PATHs entre outras coisas.
URLIB	Serviços de web.

SOCKET	Serviços de socket.
TIME	Manipulação de tempo.
IPADDRESS	manipulação de endereços IP.
CGI	Interfaces baseada em REST.
XMLRPC	Procedimentos remotos.
HMAC	Handshake para autenticação.
SSL	camadas SSL para segurança.
THREADING	Manipulação de threadings.
CONFIGPARSER	Manipulação de arquivos de configuração.
LOGGING	Manipular funções de logging.
RESOURCE	Impõe limites de memória ou de CPU a programas (unix).
WEBBROWSER	Iniciar um browser web.
QRCODE	Gerador e manipulador de QRCODES.
PYSIM	Programação simbólica (à moda do Maple)
MATPLOTLIB	Gráficos em python

Capítulo 12

Pacote: Numpy

Para usar o numpy:

Importação Se quiser chamar depois os módulos como `np.modulo` importe assim

```
>>> import numpy as np
```

Se quiser chamar direto sem usar a notação de ponto, importe assim

```
>>> from numpy import *
```

Para criar matriz O elemento `array` é central em numpy:

```
a=np.array([[1,2,3],[3,4,5],[6,6,7]])
```

ou

```
a=np.array((1,2,3),(3,4,5),(6,6,7))
```

```
>>> a
```

```
array([[1, 2, 3],
       [3, 4, 5],
       [6, 6, 7]])
```

ambos jeitos devolvem. A explicação para isto é que na criação, tanto faz fornecer uma lista ou uma tupla. Deve-se salientar que operações em `arrays` são até 2 ordens de magnitude mais rápidas que idênticas operações em `lists`. (referência DEI20, pág. 240). Até 450 pacotes do mundo Python são dependentes de Numpy. (Isto é: exigem que Numpy esteja instalado antes de funcionarem).

Para indexar

`a[1][2]` é igual a 5 OU `a[1,2]` que igualmente é 5

`in` É usado para testar se um elemento está num array ou não:

```
>>> a = np.array([[1, 2, 3], [4, 5, 6]], float)
```

```
>>> 2 in a
```

```
True
```

```
>>> 0 in a
```

```
False
```

Criação de PA A função `arange()` cria uma PA, cujos termos são `a=np.arange(vl_inicial, até_vl_final, incremento)`, como em

```
a=np.arange(1,5,0.5)
```

```
>>> a
```

```
array([ 1. , 1.5, 2. , 2.5, 3. , 3.5, 4. , 4.5])
```

Atributos de ndarray Um ndarray tem os seguintes atributos:

`.ndim` = número de dimensões

`.shape` = uma tupla com as dimensões

```
>>> a
```

```
array([1, 2, 3, 4])
```

```
>>> a.shape
```

```
(4,)
```

Cópia de array Para criar um array como cópia de outro, fazer

```
s3 = s2.copy()
```

Reorganizar um array Para reorganizar um array:

```
>>> a1 = np.arange(0.0, 12.0, 1.0)
>>> a1
[ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10. 11.]
>>> a2 = a1.reshape( (2,6) )
>>> print a2
[[ 0.  1.  2.  3.  4.  5.]
 [ 6.  7.  8.  9. 10. 11.]]
```

Reorganizar in place Para reorganizar in-place

```
>>> a1
[ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10. 11.]
>>> a1.resize([2,6])
>>> a1
[[ 0.  1.  2.  3.  4.  5.]
 [ 6.  7.  8.  9. 10. 11.]]
```

Média Para retornar a média, variância, soma e produto:

```
A.mean() -> média dos valores em A
A.var() -> variancia dos valores de A
A.sum() -> soma
A.prod() -> produto
A.max() -> maior valor
A.min() -> menor valor
A.argmax() -> índice do maior valor
A.argmin() -> índice do menor valor
unique(a) -> retorna os valores únicos
              (sem repetição) de a
```

Funções Universais Para funções universais (ufuncs)

```
>>> a=[1,2,3,4]
>>> b=[0.1, 0.7, 8, 9]
>>> a
[1, 2, 3, 4]
>>> b
[0.1, 0.7, 8, 9]
>>> a+b
[1, 2, 3, 4, 0.1, 0.7, 8, 9]
>>> a=np.array([1,2,3,4])
>>> b=np.array([0.1, 0.7, 8, 9])
>>> a+b
array([ 1.1,  2.7, 11. , 13. ])
```

Broadcasting Para broadcasting (aplicar um escalar a um array)

```
>>> a
array([1, 2, 3, 4])
>>>
>>> a+3
array([4, 5, 6, 7])
>>> b*5
array([ 0.5,  3.5, 40. , 45. ])
```

multiplicação matricial Multiplicação matricial:

```
>>> a1 = np.array([[1, 2, -4], [3, -1, 5]])
>>> a2 = np.array([[6, -3], [1, -2], [2, 4]])
>>> np.dot(a1,a2)
array([[ 0, -23],
       [ 27, 13]])
```

Produto vetorial O produto vetorial é

```
>>> x = (1, 2, 0)
>>> y = (4, 5, 6)
>>> print np.cross(x, y)
[12 -6 -3] para resolver isto ache o
           determinante da matriz

i j k
1 2 0
4 5 6, com resultados em i, j e k
```

determinante Para obter o determinante de uma matriz

```
>>> m = np.array(((2,3), (-1, -2)))
>>> m
[[ 2 3]
 [-1 -2]]
>>> np.linalg.det(m)
-1.0
```

Matriz inversa É a matriz inversa

```
>>> good = np.array(((2.1, 3.2), (4.3, 5.4)))
>>> np.linalg.inv(good)
[[-2.23140496 1.32231405]
 [ 1.7768595 -0.8677686 ]]
```

Sistema de equações lineares A solução de um sistema de equações lineares por exemplo

```
2x + y = 19
x - 2y = 2
cuja solução é (x=8 and y=3):
>>> coeffs = np.array([[2,1], [1,-2]])
>>> coeffs
[[ 2 1]
 [ 1 -2]]
>>> consts = np.array((19, 2))
>>> consts
[19 2]
>>> np.linalg.solve(coeffs, consts)
[ 8. 3.]
```

Transposta

```
>>> a = np.array(range(6), float).reshape((2, 3))
>>> a
array([[ 0., 1., 2.],
       [ 3., 4., 5.]])
>>> a.transpose()
array([[ 0., 3.],
       [ 1., 4.],
       [ 2., 5.]])
```

Concatenação

```
>>> a = np.array([1,2], float)
>>> b = np.array([3,4,5,6], float)
>>> c = np.array([7,8,9], float)
>>> np.concatenate((a, b, c))
array([1., 2., 3., 4., 5., 6., 7., 8., 9.])

>>> a = np.array([[1, 2], [3, 4]], float)
>>> b = np.array([[5, 6], [7,8]], float)
>>> np.concatenate((a,b))
array([[ 1.,  2.],
       [ 3.,  4.],
       [ 5.,  6.],
       [ 7.,  8.]])
>>> np.concatenate((a,b), axis=0)
array([[ 1.,  2.],
       [ 3.,  4.],
       [ 5.,  6.],
       [ 7.,  8.]])
>>> np.concatenate((a,b), axis=1)
array([[ 1.,  2.,  5.,  6.],
       [ 3.,  4.,  7.,  8.]])
```

Funções matemáticas abs, sign, sqrt, log, log10, exp, sin, cos, tan, arcsin, arccos, arctan, sinh, cosh, tanh, arcsinh, arccosh, and arctanh.

floor, ceil, and rint give the lower, upper, or nearest (rounded) integer:

Lista de nomes definidos Para retornar uma lista de nomes definidos

```
>>> dir()
```

subpacotes Lista de subpacotes que ajudam na programação:

subpacote	Objetiv
core	basic objects
lib	additional utilities
linalg	basic linear algebra
fft	discrete Fourier transforms
random	random number generators
distutils	enhanced build, distrib improvements
testing	unit-testing utility functions
f2py	automatic wrapping of Fortran code

Geração de aleatórios Compreendidos entre 0 e 1.

```
>>> np.random.rand(3,2)
array([[ 0.14022471,  0.96360618], #random
       [ 0.37601032,  0.25528411], #random
       [ 0.49313049,  0.94909878]]) #random
```

Inteiros aleatórios Veja o exemplo

```
>>> randint(low[, high, size]) # Return random integers
    from low (inclusive) to high (exclusive)
```

Para mostrar

```
>>> import matplotlib.pyplot as plt
>>> d1 = np.random.random_integers(1, 6, 1000)
>>> d2 = np.random.random_integers(1, 6, 1000)
>>> dsums=d2+d2
>>> count,bins,ignored = plt.hist(dsums,11,normed=True)
>>> plt.show()
```

Distribuição normal Esta função retorna simulações de uma distribuição normal. Para obter $N(\mu, \sigma^2)$, basta fazer $\text{sigma} * \text{np.random.randn}(\dots) + \text{mu}$. Onde μ = média da distribuição e σ^2 é a variância. Veja-se o exemplo

```
>>> 2.5 * np.random.randn(2, 4) + 3
array([[ -4.4940,  4.0095, -1.8181,  7.2971], #random
       [  0.3992,  4.6845,  4.9939,  4.8405]]) #random
```

Cria um array 2,4 contendo simulações de 3,6.25. Lembre que $2.5^2 = 6.25$.

Distribuição binomial Cria simulações desta distribuição. Usa 2 parâmetros: **n**, que significa o número de tentativas e **p** que vem a ser a probabilidade de sucesso da distribuição ($0 \leq p \leq 1$). A função retorna para cada simulação o número de sucessos. Lembrando que a probabilidade é

$$P(N) = \binom{n}{N} p^N (1-p)^{n-N}$$

) Veja-se o exemplo:

```
>>> n=10
>>> p=0.5
>>> s=np.random.binomial(n,p,1000)
```

O exemplo acima faz 1000 simulações, cada uma jogando 10 moedas e informando a quantidade de caras em cada simulação.

Ordenação

```
>>> a = np.array([[1,4],[3,1]])
>>> np.sort(a) # sort along the last axis
array([[1, 4],
       [1, 3]])
>>> np.sort(a,axis=None) #sort flattened array
array([1, 1, 3, 4])
>>> np.sort(a,axis=0) # sort along first axis
array([[1, 1],
       [3, 4]])
```

Desvio padrão em arrays

```
>>> a = np.array([[1, 2], [3, 4]])
>>> np.std(a)
1.1180339887498949
>>> np.std(a, axis=0)
array([ 1.,  1.])
>>> np.std(a, axis=1)
array([ 0.5,  0.5])
```

12.1 Varejão

12.1.1 Para gravar arquivo de números

```
def testagra(a):
    f=open("f:/p/up2017/ofid/primos.lis","w")
    f.write(str(a)[1:-1])
    f.close()
```

12.1.2 Para ler o arquivo acima

e converter os números em uma lista

```
def leiapri():
    f=open("f:/p/up2017/ofid/primos.lis","r")
    a=f.read()
    pri=list(map(int,a.split(", ")))
```

12.1.3 Para evitar a assunção de formato E na impressão

Muitas vezes é desagradável ter que lidar com matrizes impressas no formato exponencial. Para evitar isso, pode-se emitir o comando

```
np.set_printoptions(suppress=True)
```

Outra possibilidade é criar a função

```
import numpy as np
def impmat(m,t,d):
    return ('\n'.join([''.join(['{:'+str(t)+'.'+str(d)+'f']). \
        format(item) for item in row]) for row in a]))
a=np.array([[0.0001,23456.8],[33,2]])
print(impmat(a,12,8))
```

Outra possibilidade que nem sempre funciona legal, mas que é interessante é

```
print(a.round(t)) onde t é a quantidade de casas após a virgula
```

Capítulo 13

Pacote: Sympy

O Symbolic Python é um pacote Python que implementa um sistema de computação simbólica (CAS=Computer Algebra System) no ambiente Python. Para entender o que é um sistema deste tipo, acompanhe:

computação numérica `sqrt(8)` é igual a 2.8284271247461903

computação simbólica `sqrt(8)` é igual a $2\sqrt{2}$

Sendo que aqui tem-se uma constatação: ir do numérico para o simbólico não é possível, mas ir do simbólico para o numérico é fácil (neste caso basta fazer `sqrt(8).evalf()` cuja resposta será a mesma de cima – até porque é calculada do mesmo jeito: 2.8284271247461903).

Fica evidente que a computação simbólica abrange a computação numérica mas é muito (muito !) mais do que ela.

As vantagens de usar o sympy versus o Maple (por exemplo) são pelo menos duas: sympy é de graça (Maple custa 1.000 dólares) e por ser em python admite que você use tudo o que já sabe de python e também permite usar os mais de 135.000 pacotes também de graça.

Instalação e uso O sympy precisa ser instalado. Na versão para windows, você deve localizar o diretório scripts e nele fazer `pip install sympy` em um computador que tenha acesso à Internet. Se quiser usar winpython, nele o sympy já está pré-instalado. Num ou noutro, precisa importar o pacote. Faça `from sympy import *` ou algum comando equivalente.

symbols Como você está num ambiente 100% python, precisa dizer que algumas variáveis são simbólicas. Isto é feito pelo método `symbols` como em

```
from sympy import symbols
>>> x,y = symbols('x y')
>>> expr = x+2*y
>>> expr
x + 2*y
>>> expr + 1
x + 2*y + 1
>>> expr - x
2*y
>>> x*expr
x*(x + 2*y)
>>> expand(x*expr)
x**2 + 2*x*y
>>> factor(x**2 + 2*x*y)
x*(x + 2*y)
```

O sympy pode fazer todo o tipo de computação simbólica, como simplificar, calcular derivadas, integrais e limites, resolver equações, trabalhar com matrizes, e muito mais, tudo de maneira simbólica. Ele inclui módulos para plotagem, impressão, geração de código, física, estatística, combinatória, teoria dos números, geometria, lógica, etc. Vejamos um exemplo para calcular: a derivada de

$$\sin(x) \times e^x$$

$$\int ((e^x \times \sin(x) + e^x \times \cos(x)))dx$$

e também

$$\int_{-\infty}^{\infty} \sin(x^2)dx$$

```

>>> from sympy import *
>>> x, t, z, nu = symbols('x t z nu')
>>> init_printing(use_unicode=True)
>>> diff(sin(x)*exp(x), x)
      x      x
e .sin(x) + e .cos(x)
>>> integrate(exp(x)*sin(x) + exp(x)*cos(x), x)
      x
e .sin(x)
>>> integrate(sin(x**2), (x,-oo, oo))

```

Substituição

```

>>> expr = cos(x)+1
>>> expr.subs(x,y)
cos(y) + 1
>>> expr.subs(x,0)
2
>>> expr = x**y
expr = expr.subs(y, x**y)
>>> expr
x**(x**y)
>>> expr = x**3 + 4*x*y - z
>>> expr.subs([(x,2),(y,4),(z,0)])
40

```

Sympify Não confundir com **simplify**. Transforma um string em uma expressão SymPy.

```

a = "x**2 + 3*x - 1/2"
ex = sympify(a)
ex.subs(x,2)
19/2

```

evalf Permite achar o valor em ponto flutuante de uma expressão

```

>>> ex = sqrt(8)
>>> ex.evalf(20) # 20=tamanho
2.8284271247461900976
>>> ex2 = cos(2*x)
>>> ex2.evalf(subs={x:2.4})
0.0874989834394464

```

Simplify simplificação de expressões.

```

>>> simplify(sin(x)**2 + cos(x)**2)
1
>>> simplify((x**3+x**2-x-1)/(x**2+2*x+1))
x - 1

```

Expand Expande expressões.

```

>>> expand((x+1)**2)
2
x + 2.x + 1
>>> expand((x+2)*(x-3))
2
x . x - 6

```

Factor Fatora expressões.

```

>>> factor(x**2*z + 4*x*y*z + 4*y**2*z)
2
z.(x + 2.y)

```

Outros comandos `collect` = coleciona potências comuns
`cancel` = transforma polinômio na forma p/q
`apart` = executa uma decomposição em frações parciais
`trigsimp` = simplificação trigonométrica
`expand_trig` = expansão trigonométrica
`factorial`, `binomial`, `gamma`, `power`, `powers`, ...

Derivadas podem ser chamadas com `diff(expressao)` e com `Derivative(expressao)`.

```
>>> diff(cos(x),x)
-sin(x)
```

Integrais o comando que integra é `integrate`. O comando `Integral` serve para representar a integral e pode ser calculado depois com `x.doit()`. Atente que `integrate` é com minúscula e `Integral` é com maiúscula. Acompanhe

```
>>> x = Integral(log(x)**2,x)
>>> x
/
|  2
| log (x) dx
/
>>> x.doit()
  2
x.log (x) - 2.x.log(x) + 2.x
>>>
sin(x)
```

Limites

```
>>> limit(sin(x)/x, x, 0)
1
```

Solve Em Sympy a igualdade em uma equação é dada por `Eq` como em $x^2 = 1 \Rightarrow \text{Eq}(x^2,1)$ e para resolver esta equação escreve-se

```
>>> solveset(Eq(x**2,1), x)
{-1, 1}
```

Note que isto pode ser escrito como `solveset(Eq(x**2-1,0), x)` e neste caso pode ser também `solveset(x**2-1, x)` (já que é igual a zero). Para resolver um sistema de equações o comando é `linsolve`, veja

```
>>>linsolve([x+y+z-1,x+y+2*z-3],[x,y,z])
{(-y-1, y, 2)}
>>>linsolve(([1,1,1,1],[1,1,2,3]),[x,y,z])
{(-y-1, y, 2)}
```

Para sistemas não lineares, o comando é `nonlinsolve`.

```
>>>nonlinsolve([x*y-1,x-2],x,y)
{(2, 1/2)}
```

Para equações diferenciais, use `dsolve`.

Matrizes Para fazer matrizes em Sympy use o objeto `Matrix`, como em `Matrix([[1, -1], [3, 4], [0, 2]])`. Matrizes não são imutáveis, podendo ser alteradas.

Para a matriz `M`, sua forma é `M.shape`. Para achar a inversa fazer `M**-1`. Para achar a transposta, faça `M.T`. Para o determinante, use `M.det`.

```
>>> M=Matrix([[1,2,3],[10,20,51],[8,-1,9]])
>>> M
+1  2  3 +
|      |
|10 20 51|
|      |
+8  -1  9 +
>>> M**-1
```

```

+ 11      +
| --    -1/17    2/17 |
| 17      |
|106     |
|---    -5/119   -1/17|
|119     |
|        |
|-10     |
|--    1/21     0    |
+ 21      +
>>> M.T
+1  10  8 +
|      |
|2  20 -1|
|      |
+3  51  9 +

```

Segue a documentação tratando de teoria dos números (basicamente as tratativas referentes a números primos, depois criptografia, depois matemática concreta, depois geometria, depois integração, lógica, séries, ... a documentação do pacote tem (em maio de 2018) 1.878 páginas.

13.0.1 Exercício

1. Resolva (usando `solveset`) a equação:

$$3x^3 + 4x^2 + 32x + 256 = 0$$

no domínio dos números reais.

2. Resolva a integral definida a seguir,

$$\int_6^8 (3 \times \sin(x))$$

de maneira numérica, usando o comando `Integral` de SymPy nos intervalos dados e depois calcule o valor numérico da integral usando o método `as_sum` com 50 intervalos e tomando como parâmetro o ponto médio de cada intervalo: (Forneça a resposta com 3 casas decimais)

```

>>> init_printing(use_unicode=True)
>>> e = Integral(...funcao..., (x,a,b))
>>> e
>>> e.as_sum(50, 'midpoint').evalf()

```

1	2
---	---

13.1 Um exemplo de plotagem de uma função

A seguir, a apresentação de uma função e suas duas derivadas (a primeira e a segunda) com as funções, calculadas pelo sympy, resolvidas por ele e desenhadas pelo matplotlib. No script a seguir, o usuário tem que alterar 3 coisas:

- a função f lançada na forma de texto
- o valor vi indicando a coordenada x inicial da plotagem
- o valor vf indicando a coordenada x final da plotagem.

O programa emite as 3 funções (f , df/dx e d^2f/dx^2), as raízes das 3 funções e finalmente o gráfico gerado, no qual a cor azul é usada para f , vermelho para df/dx e verde para d^2f/dx^2 . Veja-se um exemplo:

```

# funcao para desenhar uma função genérica e suas 2 derivadas
import matplotlib.pyplot as plt
import numpy as np
from sympy import *
x, y, z, t = symbols('x y z t')

```

```

k, m, n = symbols('k m n', integer=True)
f, g, h = symbols('f g h', cls=Function)
init_printing()
#f=x**3-2*x**2+x
f=sin(x)
g=diff(f,x)
h=diff(g,x)
print('f          = ',end='')
pprint(f)
print('df/dx     = ',end='')
pprint(g)
print('d2f/dx2   = ',end='')
pprint(h)
print('raizes f      = ',solve(f,x))
print('raizes df/dx  = ',solve(g,x))
print('raizes d2f/dx2 = ',solve(h,x))
vi=-1
vf=4
x1=np.arange(vi,vf,((vf-vi)/100))
y1=[0.0]*100
y2=[0.0]*100
y3=[0.0]*100
for i in range(100):
    y1[i]=f.subs(x,x1[i])
    y2[i]=g.subs(x,x1[i])
    y3[i]=h.subs(x,x1[i])
plt.plot(x1,y1,'b-',label='função original')
plt.plot(x1,y2,'r-',label='derivada primeira')
plt.plot(x1,y3,'g-',label='derivada segunda')
plt.grid()
plt.legend()
plt.title("Comportamento da função")
plt.show()

```

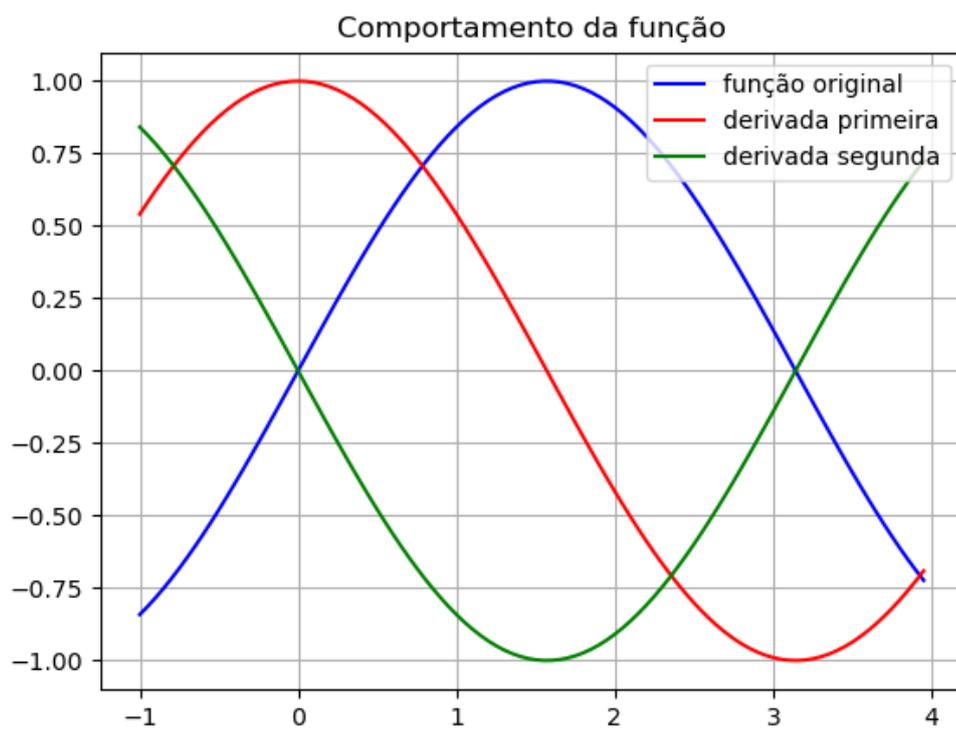
A resposta literal foi

```

f          = sin(x)
df/dx     = cos(x)
d2f/dx2   = -sin(x)
raizes f      = [0, pi]
raizes df/dx  = [pi/2, 3*pi/2]
raizes d2f/dx2 = [0, pi]

```

e o desenho gerado foi



13.1.1 Resposta

-4 3.317

Capítulo 14

Pacote: Astropy

14.1 Padrão FITS

FITS é uma sigla que significa *Flexible Image Transport System* e é um padrão aberto que define um formato de arquivo digital usado para armazenar, transmitir e processar imagens (originalmente) científicas. O padrão nasceu na área de astronomia justamente para permitir o intercâmbio de imagens digitais obtidas em telescópios de todas as naturezas possíveis. O arquivo é suficientemente genérico para acomodar muitos dados científicos além da imagem em si. Por exemplo, é comum haver dados de fotometria, informações de calibração, filtros usados além de muitas informações sobre a imagem em si, o que chamamos metadados. O padrão FITS surgiu em 1981 e evoluiu gradualmente desde então, a versão mais recente é a 3 de 2008. O mote do FITS é *once FITS, always FITS* e ele tem a pretensão de ser legível e manuseável mesmo depois que o software e o hardware usados para criar a imagem original não existam mais. Repare que este requisito inviabiliza o uso de formatos proprietários (GIF) bem como aqueles dependentes de alguma tecnologia específica (JPG, BMP, RAW, PDF, etc). O formato ganhou visibilidade quando foi escolhido pela Biblioteca do Vaticano para disponibilizar imagens de documentos daquele lugar, talvez a mais importante biblioteca de documentos antigos do mundo, repleta de manuscritos dos séculos V, VIII, XI e assim por diante. Suas principais características:

Não proprietário O formato FITS é mantido pela comunidade científica internacional, em particular a comunidade de astrônomos e astrofísicos e como tal ele não é formato proprietário. Atualmente a especificação é mantida pelo Grupo de Trabalho FITS da IAU, União Internacional de Astronomia.

Bem documentado O formato FITS está completamente documentado na Referência FITS, livremente disponível. A versão 3 foi publicada no *Astronomy and Astrophysics* volume 524 de dezembro de 2010.

Largamente adotado O formato é comumente usado pelos astrônomos. Há muitas aplicações de software para criar e ver os arquivos, além de permitir a importação e exportação em outros formatos também usuais.

Transparência Uma grande vantagem é a simplicidade do formato. Nada é comprimido (o que introduziria grande risco em caso da corrupção de um único bit...) e os dados são guardados ora em ASCII puro, ora em formatos de codificação padronizados.

Auto contido Todas as informações necessárias para representar corretamente a imagem (na tela ou na impressora) estão incluídas dentro do arquivo. Links externos ou fontes de recursos externos não são permitidos.

Independência de dispositivo Não há nenhuma dependência a algum hardware ou software específicos em nenhum ponto do arquivo. Esta característica é muito importante na compatibilidade para a frente, pois não temos idéia do que vem por aí.

14.1.1 Formato

O arquivo é formado por conjuntos de header + dados, tantos quantos houver interesse. O primeiro conjunto é chamado primário e é obrigatório. Quando só tem o primário o arquivo é conhecido como Arquivo FITS básico e quando tem uma ou mais extensões seguindo o conjunto primário é chamado de Arquivo FITS multi-extensão. Cada conjunto (chamado na documentação de *FITS structure*) é formado por um número inteiro de blocos FITS, cada um com 2880 bytes e escrito em ASCII puro ($2880 = 80 \times 36$). Todos estes blocos ficam juntos, antes dos dados. Por ASCII puro entende-se caracteres de 32 até 126.

O bloco de header contém uma série de palavras chaves seguida de seu valor. Assim, cada header tem espaço para 36 palavras chave. O último bloco de header deve conter a palavra chave **END** que marca o fim lógico do header. O que sobrar de espaço depois, deve ser preenchido com espaços.

O conjunto de dados que vem depois, (se presente) deve consistir de um arranjo (array) de dados de 1 a 999 dimensões (isto é definido pela palavra chave **NAXIS** O array inteiro é representado por um stream contínuo de bits que começa no primeiro bit do primeiro bloco de dados. Cada dado consiste de um número fixo de bits como determinado na palavra

chave **BITPIX**. O índice do eixo 1 é o que varia mais rapidamente, depois o do eixo 2, e assim por diante. O último eixo (**NAXIS**) é o que varia mais lentamente. Não há espaço ou qualquer caractere especial entre o último valor em uma linha ou plano e o primeiro valor da próxima linha ou plano. A contagem começa em 1. Veja na figura como é isso:

```
A(1, 1, ..., 1),
A(2, 1, ..., 1),
...,
A(NAXIS1, 1, ..., 1),
A(1, 2, ..., 1),
A(2, 2, ..., 1),
...,
A(NAXIS1, 2, ..., 1),
...,
A(1, NAXIS2, ..., NAXISm),
...,
A(NAXIS1, NAXIS2, ..., NAXISm)
```

Se o bloco de dados não preenche o último bloco ele deve ser preenchido com bits zero.

O bloco de header é formado por registros de 80 caracteres (36 por bloco) e eles consistem de uma palavra chave, um indicador de valor (se necessário), um valor opcional e um comentário opcional. As palavras podem aparecer em qualquer ordem, com poucas exceções. A palavra chave deve ter até 8 posições, alinhada à esquerda e com brancos à direita, sempre em letras maiúsculas. As posições 9 e 10 podem ter "=_". A seguir o valor da palavra chave e opcionalmente um comentário. Entre o valor e o comentário deve-se escrever "_/". Em todo este conteúdo (inclusive comentários) deve-se usar apenas os caracteres ASCII de 32 até 126 – hexadecimal **20** até **7E**. Variáveis de valor lógico devem ser representadas por **T** ou **F**.

Eis as principais palavras chave:

SIMPLE Deve conter um valor verdade (**T**) se este arquivo obedece ao padrão FITS.

BITPIX Contém um inteiro e indica o número de bits que representa cada valor no array de dados. Os valores possíveis são: 8, 16, 32, 64 – para inteiros – e -32 ou -64 para ponto flutuante.

NAXIS Inteiro não negativo não maior que 999 indicando quantos eixos tem o array de dados.

NAXISn Esta palavra deve estar presente para todos os valores de n de 1 até **NAXIS**. Indica o tamanho da dimensão do array de dados.

END Indica final do cabeçalho. Esta palavra chave não tem valor associado

Veja-se um exemplo disto:

```
SIMPLE = T / file does conform to FITS standard
BITPIX = 16 / number of bits per data pixel
NAXIS = 2 / number of data axes
NAXIS1 = 250 / length of data axis 1
NAXIS2 = 300 / length of data axis 2
OBJECT = 'Cygnus X-1'
DATE = '2008-10-27'
END
```

O número total de bits do array de dados primário, sem contar o preenchimento eventualmente necessário para completar o bloco de 2880 caracteres, é $N_{bits} = |BITPIX| \times (NAXIS1 \times NAXIS2 \times \dots \times NAXISm)$

14.1.2 A cor no arquivo FITS

Paradoxalmente, a cor não tem um significado fixo no arquivo FITS. Ela pode (deve) ser livremente interpretada numa negociação entre gravador e leitor. O formato lava as mãos sobre isso. Para imagens coloridas do uso no nosso dia a dia, pode-se fazer a seguinte combinação: true color (8 bits=0..255) por canal R, G, B e mais um quarto canal de transparência. Total 32 bits por pixel. Neste caso, pode-se usar o inteiro de 32 bits no registro do arquivo. Se uma maior precisão de cor é exigida (como por exemplo na captura de pergaminhos e iluminuras da Biblioteca Vaticana), pode-se fazer: 16 bits por canal R, G ou B chegando portanto a 65.536 níveis em cada cor, e mais um canal de transparência com o mesmo nível de valores. Total 64 bits por pixel, usando-se neste caso os inteiros de 64 bits. Ou pode-se definir qualquer outra combinação, eventualmente usando-se outros esquemas de cor (como CMYB, por exemplo). O que resta importante no formato, é que cada valor sempre tem uma configuração que é múltipla de 8 bits, ou seja não há aqui os problemas de truncamento, padding ou stuffing bits de outros formatos.

14.1.3 Como ver arquivos FITS

Existe uma grande variedade de visualizadores de arquivos FITS, e como tudo no padrão, são livres para download e instalação. Um programa que é bem completo e funciona inclusive como editor para este tipo de arquivo é o FV, que tem versões para plataformas windows, unix e Mac. Seu site é <https://heasarc.gsfc.nasa.gov/fv/>. Vale uma visita. Outro é DS9, que pode ser baixado em <http://ds9.si.edu/site/Home.html>.

Especial referência deve ser feita em relação à linguagem Python. Usando a vocação desse ambiente, (106.000 pacotes em mar/17), há um pacote completo chamado ASTROPY que dá toda a funcionalidade para operar magnificamente com arquivos FITS. Mais detalhes em www.astropy.org, onde há um tutorial muito bom.

Para manusear este tipo de arquivo deve-se usar pelo menos 3 ferramentas:

- Um editor ASCII convencional (NOTEPAD++ por exemplo) para estudar o bloco de header
- Um editor de formato binário por exemplo wxHexEditor ou então o site <http://www.onlinehexeditor.com/>. Este último funciona online e não exige instalação de software. Aqui examina-se os blocos de dados do arquivo FITS
- Um visualizador de arquivos FITS como o FV ou DS9 que permitem examinar completamente o arquivo

Capítulo 15

Pacote: Pygames e Turtle

Hello world com Pygames

Seja o seguinte programa de hello world

```
background_image_filename = 'c:\python\pedrofepo.jpg'
mouse_image_filename = 'c:\python\selo_coxa.jpg'
import pygame
from pygame.locals import *
from sys import exit
pygame.init()
screen = pygame.display.set_mode((1200, 800),0,32)
pygame.display.set_caption("hello")
background = pygame.image.load(background_image_filename).convert()
mouse_cursor = pygame.image.load(mouse_image_filename).convert_alpha()
while True:
    for event in pygame.event.get():
        if event.type == QUIT:
            pygame.quit()
            exit()
        screen.blit(background, (0,0))
        x, y = pygame.mouse.get_pos()
        x-= mouse_cursor.get_width() /2
        y-= mouse_cursor.get_height() /2
        screen.blit(mouse_cursor, (x,y))
        pygame.display.update()
```

Este programa carrega duas imagens (a segunda com transparência), cria um ciclo eterno e nele deixa passar todos os eventos (é o que faz `event in pygame.event.get():`). Se o evento for `QUIT`, isto significa que o usuário teclou no 'X' que fecha a janela e portanto o programa deve ser encerrado. Senão, o programa desenha a tela principal começando em 0,0 (que é o canto esquerdo superior). Daí, ele pega a posição do cursor nas variáveis x,y (de uma vez só, olha aí o modernismo de python). Calcula um novo x,y (não entendi o porque do -, mas enfim), e desenha o cursor nessa nova posição. Finalmente, o conjunto todo é atualizado através do comando `pygame.display.update()`.

tratando eventos

O programa a seguir lista os eventos à medida em que eles ocorrem

```
import pygame
from pygame.locals import *
from sys import exit
pygame.init()
SCREEN_SIZE = (800, 600)
screen = pygame.display.set_mode(SCREEN_SIZE, 0, 32)
while True:
    for event in pygame.event.get():
        if event.type == QUIT:
            pygame.quit()
            exit()
        print(event)
        pygame.display.update()
```

Veja-se a seguir um programa de manuseio de listas (similar ao da pág 118-119 de [Men10]).

```
V=[1,4,5,6,77,"aa",8,45,46,28,33,34,39]
P=[]
I=[]
for e in V:
    x=str(e)
    print(x)
    print(x.isnumeric())
    if x.isnumeric():
        if e%2 == 0:
            P.append(e)
        else:
            I.append(e)
    else:
        print("deu xabu")
print("Pares",P)
print("Impares",I)
```

```
>>> ... executando ...
Pares [4, 6, 8, 46, 28, 34]
Impares [1, 5, 77, 45, 33, 39]
```

Criando um dicionário:

```
def criahash():
    import random
    x = range(100)
    som = 0
    di = {}
    for i in x:
        aa = random.randint(1,1000)
        print("%d - %d\n"%(i+1,aa))
        som = som + aa
        di[aa]=i+1
    som= som / 100
    print("media final = %8.2f" % som)
    print(di)
```

Implementando os números de fibonacci usando um dicionario para guardar o já calculado

```
def fibona(n):
    # supoe um difibo = {0:1, 1:1}
    if n in difibo:
        return difibo[n]
    else:
        novo = fibona(n-1) + fibona(n-2)
        difibo[n] = novo
        return novo
```

Brincando com o Logo

```
import turtle,random
# x=10
# y=0
t=turtle.Pen()
turtle.bgcolor('black')
cor=['red','blue','green','yellow','pink','aquamarine','tomato','ivory','magenta']
for x in range(10,400,5):
    t.width(random.randint(1,3))
    t.forward(x)
# t.circle(x)
    t.left(90)
# x=x+random.randint(1,5)
# x=x+5
```

```
# y=y+1
# t.pencolor(cor[y%9])
  t.pencolor(cor[random.randint(0,7)])
```

Como importar um módulo escrito alhures (por exemplo, o módulo `cpf.py` que está em `f:\p\n\024`. Abra uma sessão python *as usual*. Daí emita os seguintes comandos

```
import sys
```

Daí peça para ver se deu certo, pedindo para ver a variável `sys.path`. Ele deve responder com a lista de diretórios de path. Emita o comando

```
sys.path.append('f:\p\n\024')
```

Se você pedir para olhar a variável acima, verá que o novo diretório foi incluído sem problemas. Agora, emita o comando

```
from cpf import cpf
```

E agora pode usar a função `cpf` definida dentro daquele módulo fazendo

```
aa=cpf([1,1,1,2,2,2,3,3,3])
aa
```

Para criar uma lista com n zeros, fazer

```
lis = [0] * 33 # cria uma lista com 33 zeros
```


Capítulo 16

Pacote: Matplotlib

Este é um dos pacotes disponíveis no mundo Python para elaborar melhor as saídas na forma gráfica. Existem outros, vale a pena uma pesquisa no PyPi (Python Package Index) .

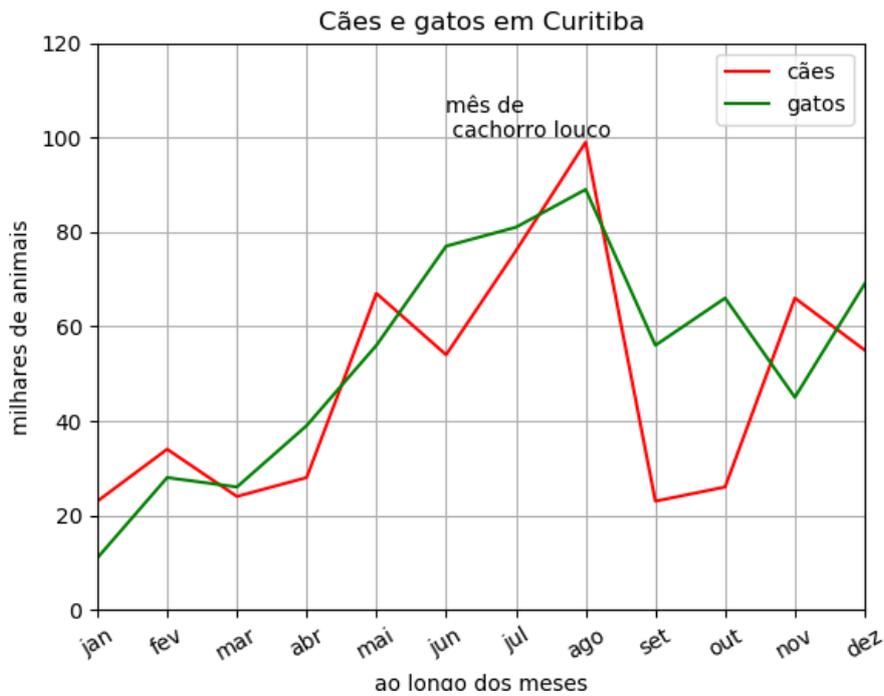
Pacote maneiro para fazer desenhos e manipulações de imagens em Python. O pacote a ser instalado é o matplotlib (pip install matplotlib) e ele exige um monte de dependências (o python, numpy, além de inúmeras outras).

16.1 Modos de uso

Há 2 modos de uso do pacote: usar a interface procedural `pyplot` ou usar a API nativa da matplotlib, esta mais orientada a objetos. A primeira é constituída por métodos (como `xlabel`, `legend`, etc) e elas podem ser usadas de duas maneiras: se chamadas sem argumentos, devolvem o valor atual do parâmetro e se se chamadas com parâmetros elas atualizam este valor. Veja a seguir uma chamada usando este método:

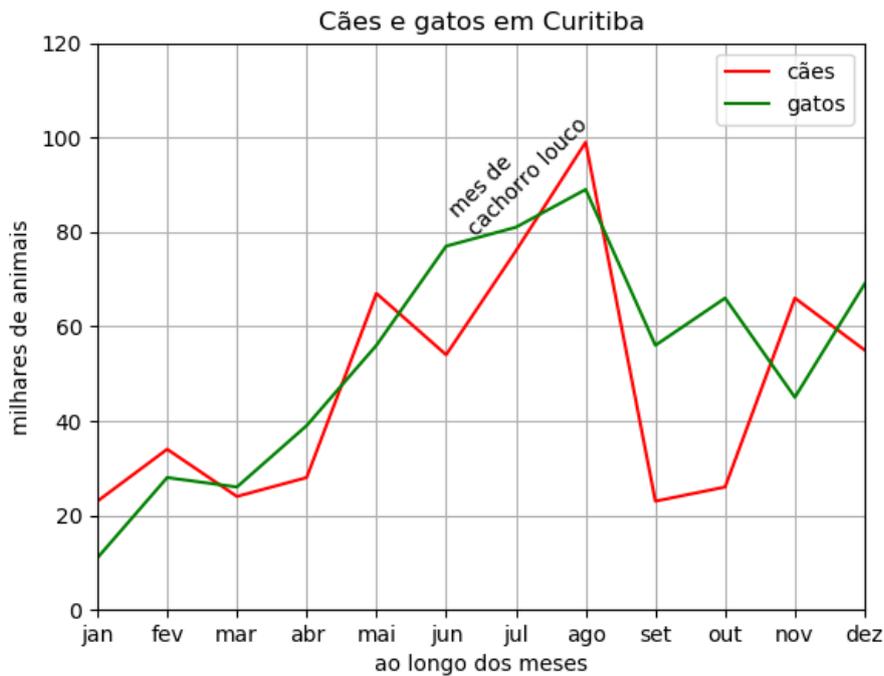
16.2 Para criar um gráfico simples, via métodos

```
# caogato
import matplotlib.pyplot as plt
import numpy as np
def caogato():
    ca=[23,34,24,28,67,54,76,99,23,26,66,55]
    ga=[11,28,26,39,56,77,81,89,56,66,45,69]
    plt.plot(ca,'r-',label='cães')
    plt.plot(ga,'g-',label='gatos')
    plt.legend()
    plt.title('Cães e gatos em Curitiba')
    plt.xlabel('ao longo dos meses')
    plt.ylabel('milhares de animais')
    plt.axis([0,11,0,120])
    plt.xticks(np.arange(12),('jan','fev','mar',
    'abr','mai','jun','jul','ago','set','out',
    'nov','dez'),rotation=30)
    plt.text(5,100,'mês de\n cachorro louco')
    plt.grid()
    plt.savefig('c:/p/python/caogato.png')
    plt.show()
caogato()
```



16.3 O mesmo gráfico, via API

```
#caogato2
import matplotlib.pyplot as plt
import numpy as np
def caogato2():
    ca=[23,34,24,28,67,54,76,99,23,26,66,55]
    ga=[11,28,26,39,56,77,81,89,56,66,45,69]
    fig,axe=plt.subplots()
    axe.plot(ca,'r-',label='cães')
    axe.plot(ga,'g-',label='gatos')
    axe.legend()
    axe.set_title('Cães e gatos em Curitiba')
    axe.set_xlabel('ao longo dos meses')
    axe.set_ylabel('milhares de animais')
    axe.axis([0,11,0,120])
    axe.set_xticks(range(12))
    axe.set_xticklabels(['jan','fev','mar',
                        'abr','mai','jun','jul','ago','set','out',
                        'nov','dez'])
    axe.grid()
    axe.text(5,100,'mes de \ncachorro louco',rotation=45)
    plt.savefig('c:/p/python/caogato2.png')
    plt.show()
caogato2()
```



t

No caso de usar a API, cada ação corresponde a dois métodos na API. Por exemplo, no caso de `xlabel`, há o método `axe.set_xlabel` para colocar coisas lá e o método `axe.get_xlabel` para recuperar o que tem lá.

16.4 O método plot()

Serve para plotar y versus x com linhas e/ou marcadores. Vejamos alguns exemplos:

`plot(y)`

Marca os valores de y usando x os índices do array `0..n-1`. Veja um exemplo

```
def exeplo1():
    y=[12,3,50,33]
    plt.plot(y)
    plt.show()
```

Agora, x pode ser explicitamente estabelecido, através de pares x,y . Veja o exemplo:

```
def exeplo2():
    x=[8,17,33,66]
    y=[12,3,50,33]
    plt.plot(x,y)
    plt.show()
```

Um segundo conjunto de dados ($x2,y2$) pode ser desenhado junto

```
def exeplo3():
    x=[8,17,33,66]
    y=[12,3,50,33]
    x2=[0,3,7,12]
    y2=[33,88,55,77]
    plt.plot(x,y,x2,y2)
    plt.show()
```

Mas isto pode ser feito em duas chamadas a `plot`, com o mesmo resultado, como em

```
def exeplo4():
    x=[8,17,33,66]
    y=[12,3,50,33]
    x2=[0,3,7,12]
    y2=[33,88,55,77]
    plt.plot(x,y)
    plt.plot(x2,y2)
    plt.show()
```

A formatação das coisas pode ser feita com propriedades de `Line2D` ou com palavras chave. Assim, ambos os valores abaixo terão o mesmo resultado

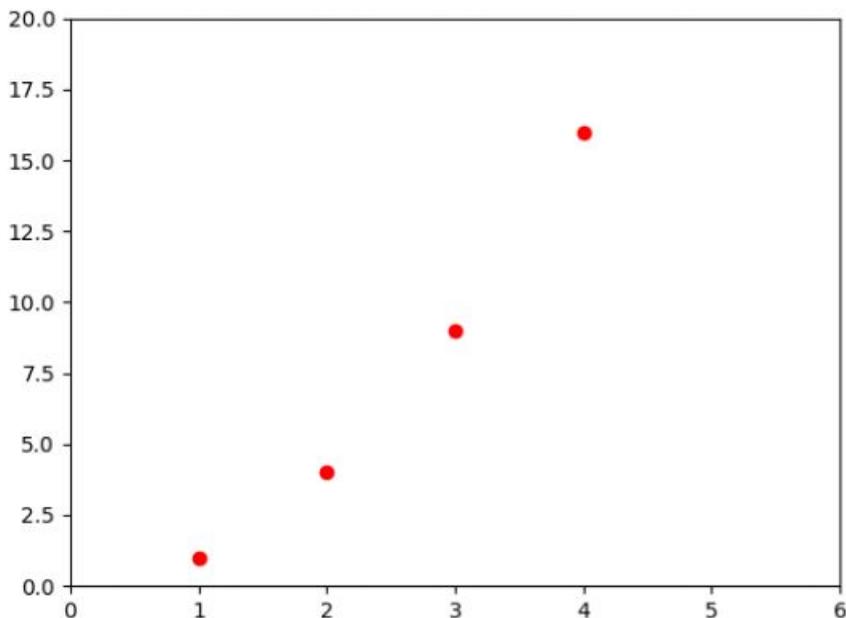
```
>>> plot(x, y, 'go--', linewidth=2, markersize=12)
>>> plot(x, y, color='green', marker='o',
        linestyle='dashed',
        linewidth=2, markersize=12)
```

As cores são: 'b'=blue, 'g'=green, 'r'=red, 'c'=ciano, 'm'=magenta, 'y'=yellow, 'k'=black e 'w'=white. Os marcadores são '.'=ponto, 'o'=bola, 'v'=triângulo, '^'=triângulo, '<'=triângulo, '>'=triângulo, 's'=quadrado, 'p'=pentágono, '*'=estrela, '+'=mais, 'x'=vezes, 'D'=diamante, 'd'=diamante estreito, '|'=linha vertical, '_'=linha horizontal, '-'=linha sólida, '-'=linha picotada, '-.'=linha ponto, ':'=linha pontilhada (tem mais...)

16.5 Um exemplo bobinho

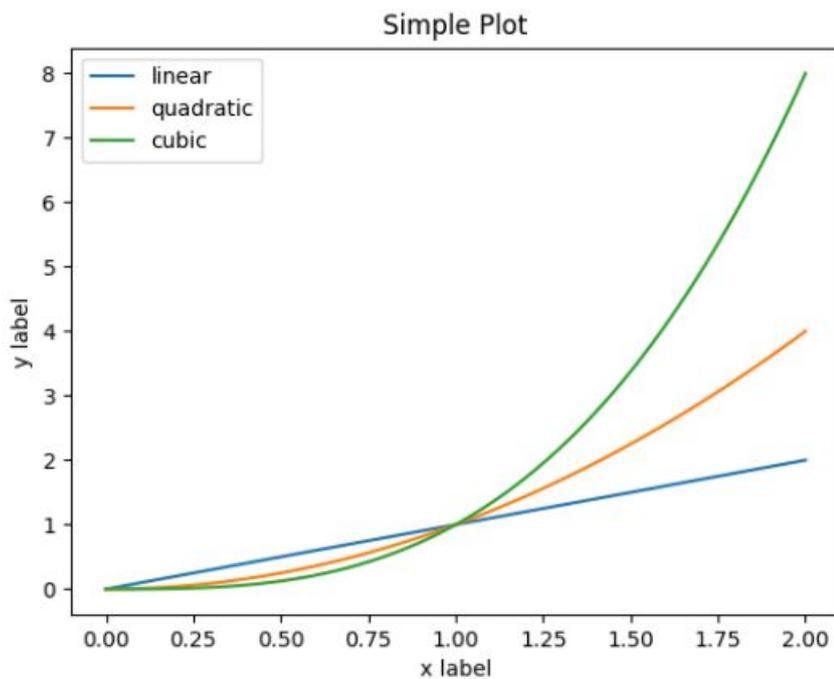
```
plt.plot([1, 2, 3, 4], [1, 4, 9, 16], 'ro')
plt.axis([0, 6, 0, 20])
plt.show()
```

`ro` significa 'red circles'. `b-` seria risco azul. `r--` é red dashes, `bs` é blue squares e `g^` é green triangles.



16.6 Um exemplo de plotagem simples

```
x = np.linspace(0, 2, 100) # pontos espaçados
plt.plot(x, x, label='linear')
plt.plot(x, x**2, label='quadratic')
plt.plot(x, x**3, label='cubic')
plt.xlabel('x label')
plt.ylabel('y label')
plt.title("Simple Plot")
plt.legend()
plt.show()
```



16.6.1 Plotando pontos

Para plotar os pontos (1,1), (2,1), (1,-1), (-2,2) e (-2,-2) deve-se fazer

```
x=[1,2,1,-2,-2]
y=[1,1,-1,2,-2]
plt.scatter(x,y)
```

Se quiser identificar os pontos faça

```
plt.text(1,1,"1")
plt.text(2,1,"2")
plt.text(1,-1,"3")
plt.text(-2,2,"4")
plt.text(-2,-2,"5")
```

Para desenhar os eixos

```
plt.vlines(x=0,ymin=-2.5,ymax=2.5)
plt.hlines(y=0,xmin=-2.5,xmax=5.5)
```

Finalmente, para conectar cada ponto à origem, faça

```
xdata=list([0, 1])
ydata=[_ for _ in xdata]
plt.plot(xdata,ydata,'b')
```

```
xdata=list([0, 2])
ydata=[_/2 for _ in xdata]
plt.plot(xdata,ydata,'r')
```

```
xdata=list([0, -2])
ydata=[abs(_) for _ in xdata]
plt.plot(xdata,ydata,'g')
```

```
xdata=list([0, -2])
ydata=[_ for _ in xdata]
plt.plot(xdata,ydata,'y')
```

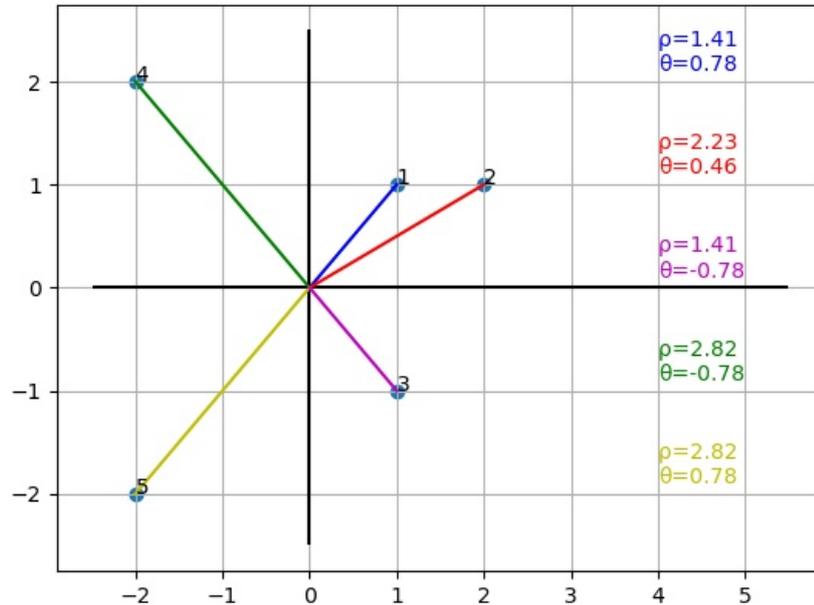
```
xdata=list([0, 1])
ydata=[-_ for _ in xdata]
plt.plot(xdata,ydata,'m')
```

Para escrever algo no texto:

```

rho=u'\u03C1'
theta=u'\u03B8'
print(rho)
plt.text(4,2.1,rho+"=1.41 \n"+theta+"=0.78",c='b')
plt.text(4,1.1,rho+"=2.23 \n"+theta+"=0.46",c='r')
plt.text(4,0.1,rho+"=1.41 \n"+theta+"=-0.78",c='m')
plt.text(4,-0.9,rho+"=2.82 \n"+theta+"=-0.78",c='g')
plt.text(4,-1.9,rho+"=2.82 \n"+theta+"=0.78",c='y')

```



Eis o resultado deste programa:

Para encerrar, uma dica preciosa: Para desenhar uma reta que vá de $P_1 = (x_1, y_1)$ até $P_2 = (x_2, y_2)$, basta fazer

```

xd=[x1, x2]
yd=[y1, y2]
plt.plot(xd,yd,'r') # r=red

```

16.7 Tortas

Eis o formato do comando `pie`

```

matplotlib.pyplot.pie(x, explode=None,
labels=None, colors=None, autopct=None,
pct-distance=0.6, shadow=False,
labeldistance=1.1, startangle=None,
radius=None, counterclock=True,
wedgeprops=None, textprops=None,
center=(0, 0), frame=False,
rotatelabels=False, hold=None, data=None)

```

As fatias estão dadas por x . Cada fatia ocupa $x/\text{sum}(x)$. Se a soma de x é menor do que 1, os valores de x indicam a fração e não são normalizados. As fatias são desenhadas em sentido anti-horário a partir do eixo x .

explode O parâmetro `explode` se não for `None` será um array de tamanho $\text{len}(x)$ indicando qual a fração do raio que cada fatia deve ser deslocada.

labels é uma sequência de strings indicando o que é cada fatia.

colors é uma sequência de cores. Se usado `None` usa-se a sequência padrão.

autopct Pode ser `None`, um string ou uma função. Diz para identificar a fatia com um número. Se for uma função ela será chamada. A `pctdistance` indica onde por este rótulo.

shadow é um booleano, indica se haverá sombra.

labeldistance o default é 1.1 e indica a distância radial onde os labels vão ser desenhados.

startangle onde começa a primeira fatia (a partir do eixo x).

radius O raio da pizza. Se *None* vale 1.

counterclock Booleano, indica o sentido antihorário ou não.

wedgeprops é um dicionário (como em `wedgeprops = {'linewidth': 3}` . Para saber mais veja a lista de argumentos do objeto `wedge`.

textprops é um dicionário com atributos de texto.

center a localização do centro da pizza (default 0,0)

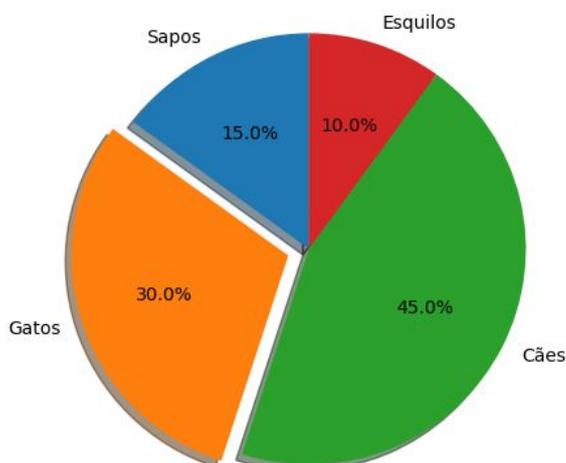
frame Booleano. Se true desenha um frame.

rotatelabels Booleano: rotaciona cada fatia no ângulo da fatia.

Retornos se chamada com *get* retorna uma lista de *wedges* outra de *texts* e de *autotexts* se *autopct* não é *None*.

startangle

```
import matplotlib.pyplot as plt
# Pie chart, where the slices will be
ordered and plotted counter-clockwise:
labels = 'Sapos', 'Gatos', 'Cães', 'Esquilos'
sizes = [15, 30, 45, 10]
explode = (0, 0.1, 0, 0) # only "explode" the
                        # 2nd slice (i.e. 'Gatos')
fig1, ax1 = plt.subplots()
ax1.pie(sizes, explode=explode, labels=labels,
        autopct='%1.1f%%', shadow=True, startangle=90)
ax1.axis('equal') # Equal aspect ratio
                #ensures that pie is drawn as a circle.
plt.show()
```



16.8 Histogramas

Desenha um histograma. Seu formato

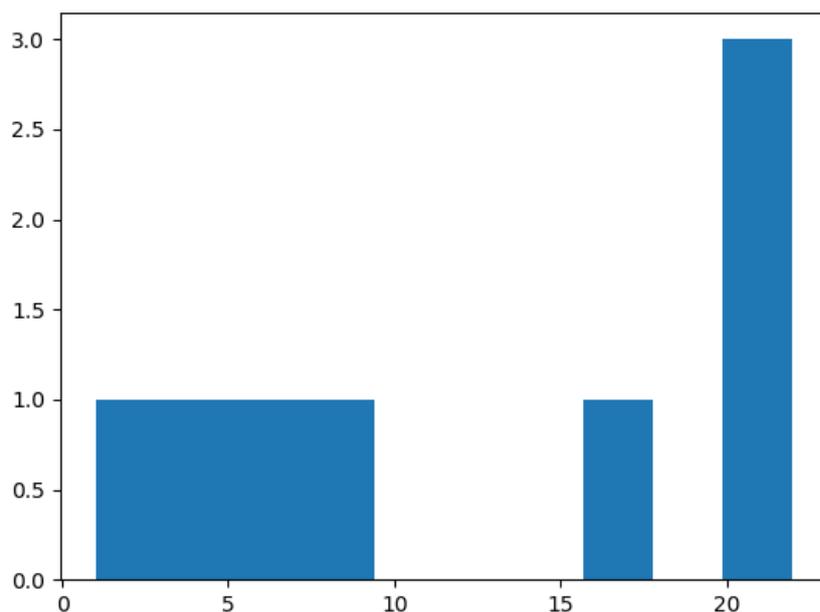
```
hist(x, bins=None, range=None,
density=None, weights=None,
cumulative=False, bottom=None,
histtype='bar', align='mid',
orientation='vertical', rwidth=None,
log=False, color=None, label=None,
stacked=False, normed=None, hold=None,
data=None, **kwargs)
```

x pode ser um array ou uma sequencia de arrays, que não precisam ter o mesmo comprimento.

Nota ***kwargs* é aqui (como em todo o matplotlib) uma lista adicional (e opcional) de palavras chave e argumentos que podem controlar mais aspectos do que está sendo pedido. Veja a documentação, a respeito.

Veja um exemplo bem bobo:

```
import matplotlib.pyplot as plt
def hist1():
    x=[1,4,7,8,16,22,22,22]
    plt.hist(x)
    plt.show()
    plt.savefig('c:/p/python/hist2.png')
hist1()
```



16.9 Um exemplo

Eis o arquivo

Median age at First Marriage, 1890–2010

Source: U.S. Bureau of the Census, www.census.gov

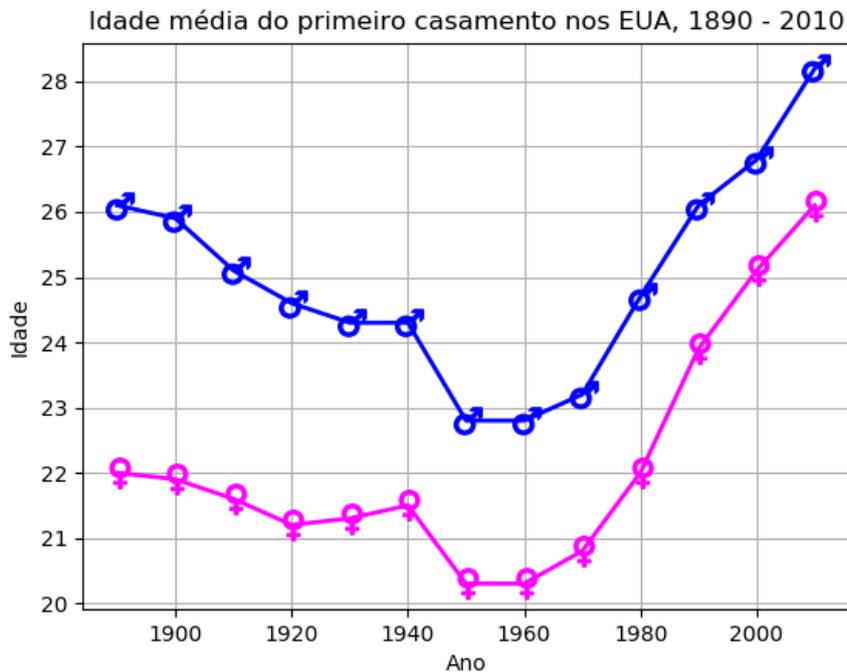
Ano	Homens	Mulheres
1890	26.1	22.0
1900	25.9	21.9
1910	25.1	21.6
1920	24.6	21.2
1930	24.3	21.3
1940	24.3	21.5
1950	22.8	20.3
1960	22.8	20.3
1970	23.2	20.8
1980	24.7	22.0
1990	26.1	23.9
2000	26.8	25.1
2010	28.2	26.1

```
import numpy as np
import matplotlib.pyplot as plt
year, age_m, age_f = np.loadtxt(
    'c:/p/python/casamentos.txt',
    unpack=True, skiprows=3)
```

```

fig = plt.figure()
ax = fig.add_subplot(111)
# Plot ages with male or female symbols as markers
ax.plot(year, age_m, marker='$\u2642$',
        markersize=14, c='blue', lw=2,
        mfc='blue', mec='blue')
ax.plot(year, age_f, marker='$\u2640$',
        markersize=14, c='magenta', lw=2,
        mfc='magenta', mec='magenta')
ax.grid()
ax.set_xlabel('Ano')
ax.set_ylabel('Idade')
ax.set_title('Idade média do primeiro
casamento nos EUA, 1890 - 2010')
plt.savefig('c:/p/python/exe1.png')
plt.show()

```

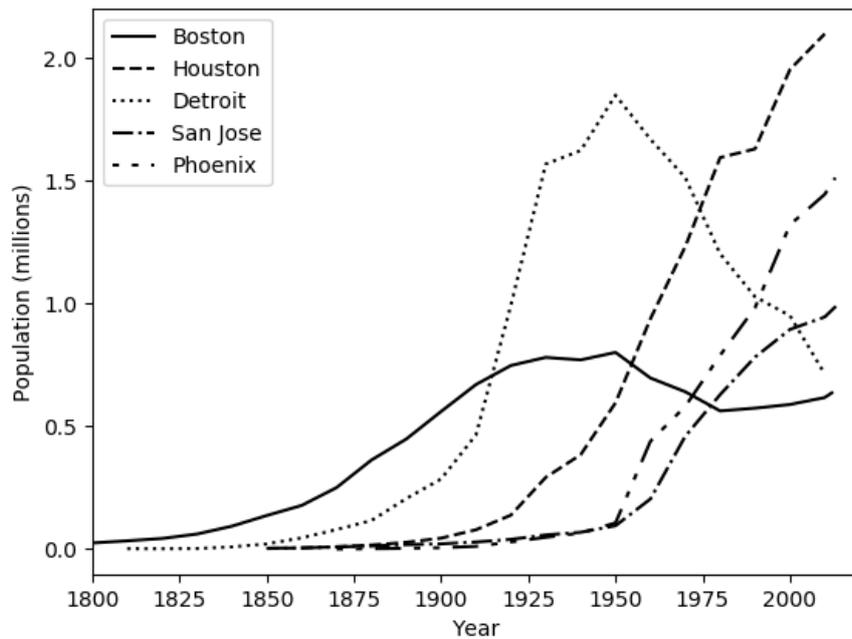


16.10 Exemplo

```

fig = plt.figure()
ax = fig.add_subplot(111)
cities = ['Boston', 'Houston',
         'Detroit', 'San Jose', 'Phoenix']
linestyles = [{ 'ls': '-' }, { 'ls': '--' },
              { 'ls': ':' }, { 'ls': '-.' },
              { 'dashes': [2, 4, 2, 4, 8, 4] }]
for i, city in enumerate(cities):
    filename = '{}.tsv'.format(city.
lower()).replace(' ', '_')
    yr, pop = np.loadtxt(filename,
unpack=True)
    line, = ax.plot(yr, pop/1.e6,
label=city, color='k', **linestyles[i])
ax.legend(loc='upper left')
ax.set_xlim(1800, 2020)
ax.set_xlabel('Year')
ax.set_ylabel('Population (millions)')
plt.show()

```

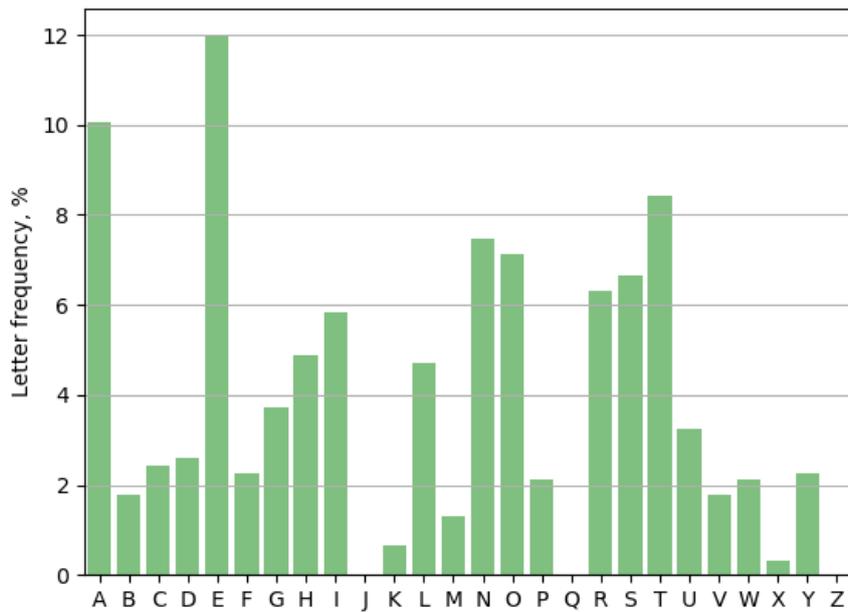


16.11 Exemplo

```

import numpy as np
import matplotlib.pyplot as plt
text_file = 'c:/p/python/mobydick.txt'
letters = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
lcount = dict([(l, 0) for l in letters])
for l in open(text_file).read():
    try:
        lcount[l.upper()] += 1
    except KeyError:
        pass
norm = sum(lcount.values())
fig = plt.figure()
ax = fig.add_subplot(111)
x = range(26)
ax.bar(x, [lcount[l]/norm * 100 for l in letters], width=0.8,
        color='g', alpha=0.5, align='center')
ax.set_xticks(x)
ax.set_xticklabels(letters)
ax.tick_params(axis='x', direction='out')
ax.set_xlim(-0.5, 25.5)
ax.yaxis.grid(True)
ax.set_ylabel('Letter frequency, %')
plt.show()

```

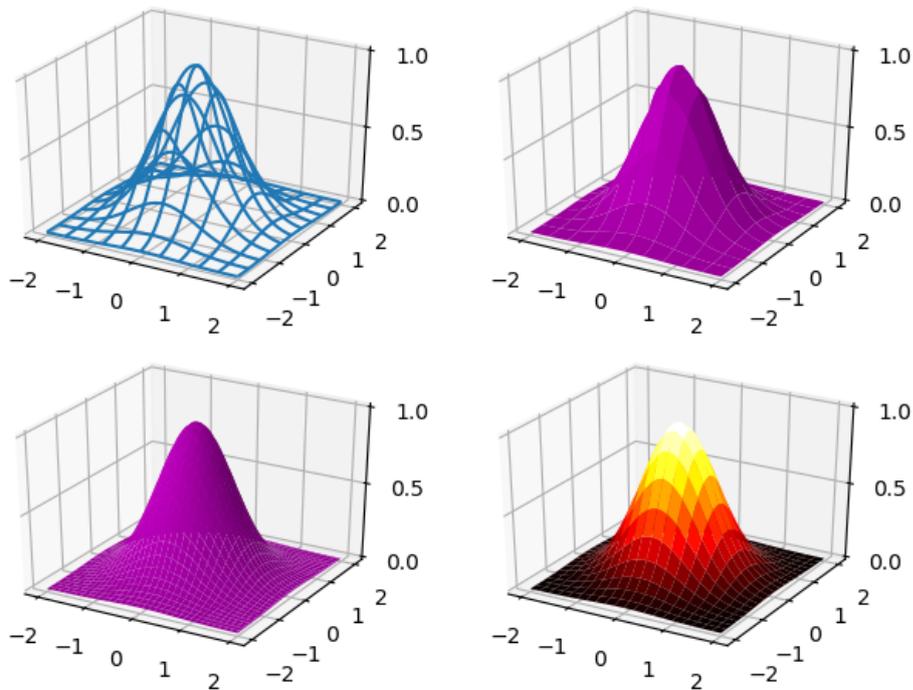


16.12 Exemplo

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.cm as cm
```

```
L, n = 2, 400
x = np.linspace(-L, L, n)
y = x.copy()
X, Y = np.meshgrid(x, y)
Z = np.exp(-(X**2 + Y**2))
```

```
fig, ax = plt.subplots(nrows=2, ncols=2, subplot_kw=
    {'projection': '3d'})
ax[0,0].plot_wireframe(X, Y, Z, rstride=40, cstride=40)
ax[0,1].plot_surface(X, Y, Z, rstride=40, cstride=40, color='m')
ax[1,0].plot_surface(X, Y, Z, rstride=12, cstride=12, color='m')
ax[1,1].plot_surface(X, Y, Z, rstride=20, cstride=20, cmap=cm.hot)
for axes in ax.flatten():
    axes.set_xticks([-2, -1, 0, 1, 2])
    axes.set_yticks([-2, -1, 0, 1, 2])
    axes.set_zticks([0, 0.5, 1])
fig.tight_layout()
plt.show()
```



16.13 Exemplo

```

import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

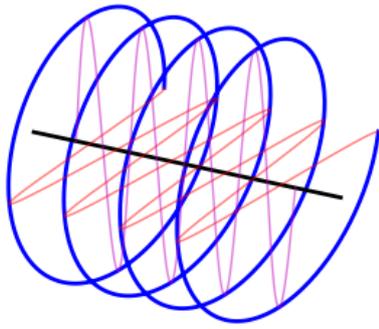
n = 1000
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Plot a helix along the x-axis
theta_max = 8 * np.pi
theta = np.linspace(0, theta_max, n)
x = theta
z = np.sin(theta)
y = np.cos(theta)
ax.plot(x, y, z, 'b', lw=2)

# An line through the centre of the helix
ax.plot((-theta_max*0.2, theta_max * 1.2), (0,0), (0,0), color='k', lw=2)
# sin/cos components of the helix (e.g. electric and magnetic field
# components of a circularly-polarized electromagnetic wave
ax.plot(x, y, 0, color='r', lw=1, alpha=0.5)
ax.plot(x, [0]*n, z, color='m', lw=1, alpha=0.5)

# Remove axis planes, ticks and labels
ax.set_axis_off()
plt.show()

```



16.14 Julia Set

```

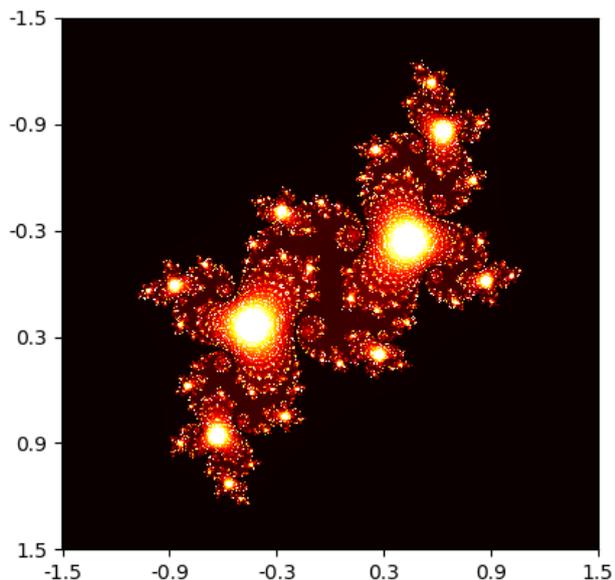
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cm as cm

# Image width and height; parameters for the plot
im_width, im_height = 500, 500
c = complex(-0.1, 0.65)
zabs_max = 10
nit_max = 1000
xmin, xmax = -1.5, 1.5
xwidth = xmax - xmin
ymin, ymax = -1.5, 1.5
yheight = ymax - ymin

julia = np.zeros((im_width, im_height))
for ix in range(im_width):
    for iy in range(im_height):
        nit = 0
        # Map pixel position to a point in the complex plane
        z = complex(ix / im_width * xwidth + xmin,
                    iy / im_height * yheight + ymin)
        # Do the iterations
        while abs(z) <= zabs_max and nit < nit_max:
            z = z**2 + c
            nit += 1
        shade = 1-np.sqrt(nit / nit_max)
        ratio = nit / nit_max
        julia[ix,iy] = ratio

fig, ax = plt.subplots()
ax.imshow(julia, interpolation='nearest', cmap=cm.hot)
# Set the tick labels to the coordinates of z0 in the complex plane
xtick_labels = np.linspace(xmin, xmax, xwidth / 0.5)
ax.set_xticks([(x-xmin) / xwidth * im_width for x in xtick_labels])
ax.set_xticklabels(['{:0.1f}'.format(xtick) for xtick in xtick_labels])
ytick_labels = np.linspace(ymin, ymax, yheight / 0.5)
ax.set_yticks([(y-ymin) / yheight * im_height for y in ytick_labels])
ax.set_yticklabels(['{:0.1f}'.format(ytick) for ytick in ytick_labels])
plt.show()

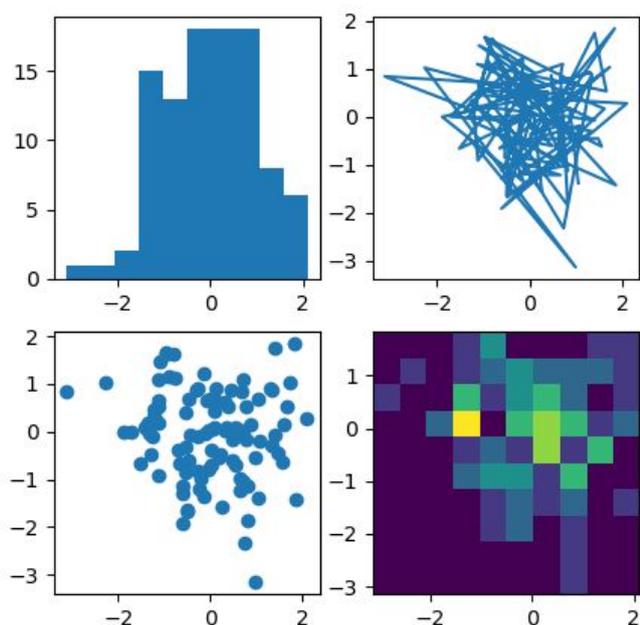
```



16.15 Um exemplo de subplot

Para escrever mais de um gráfico em um único espaço, usa-se `subplot()`. Veja a seguir um exemplo.

```
import matplotlib.pyplot as plt
import numpy as np
np.random.seed(19680801)
data = np.random.randn(2, 100)
fig, axs = plt.subplots(2, 2, figsize=(5, 5))
axs[0, 0].hist(data[0])
axs[1, 0].scatter(data[0], data[1])
axs[0, 1].plot(data[0], data[1])
axs[1, 1].hist2d(data[0], data[1])
plt.show()
```

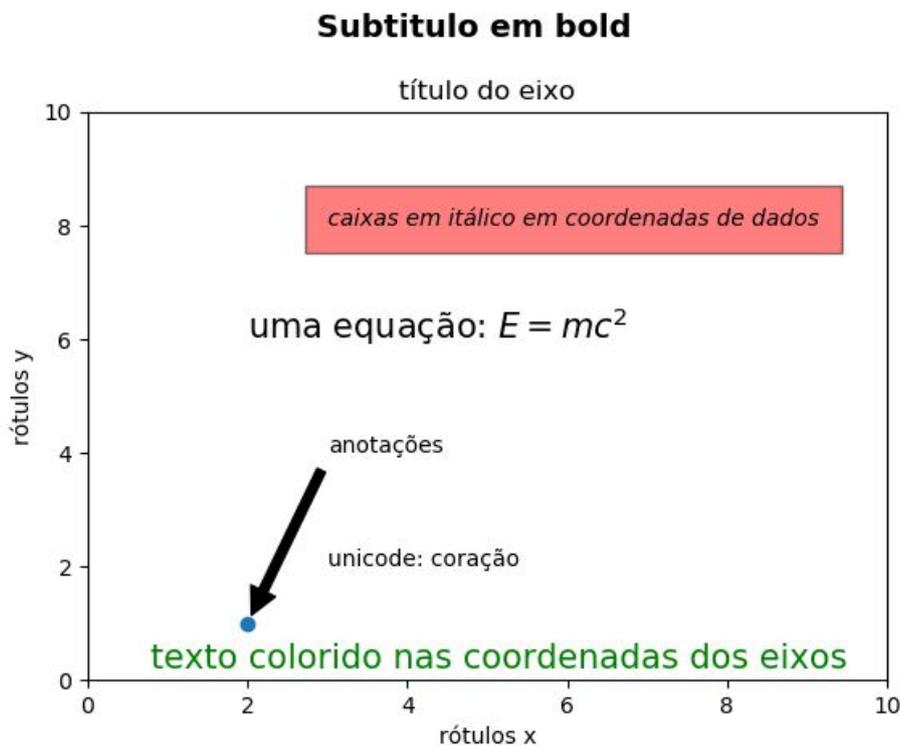


16.16 Interativo ou não

Fazendo `plt.ion()` tudo o que você manda fazer ele faz e mostra na janela. Fazendo `plt.ioff()` você faz tudo e no final tem que dar um `plt.show()` para ver o resultado.

16.17 Escrevendo texto

```
import matplotlib
import matplotlib.pyplot as plt
fig = plt.figure()
fig.suptitle('Subtítulo em bold', fontsize=14,
             fontweight='bold')
ax = fig.add_subplot(111)
fig.subplots_adjust(top=0.85)
ax.set_title('título do eixo')
ax.set_xlabel('rótulos x')
ax.set_ylabel('rótulos y')
ax.text(3, 8, 'caixas em itálico em coordenadas de
           dados', style='italic',
        bbox={'facecolor': 'red', 'alpha': 0.5, 'pad': 10})
ax.text(2, 6, r'uma equação: $E=mc^2$', fontsize=15)
ax.text(3, 2, u'unicode: coração')
ax.text(0.95, 0.01, 'texto colorido nas coordenadas
                    dos eixos',
        verticalalignment='bottom',
        horizontalalignment='right',
        transform=ax.transAxes,
        color='green', fontsize=15)
ax.plot([2], [1], 'o')
ax.annotate('anotação', xy=(2, 1), xytext=(3, 4),
           arrowprops=dict(facecolor='black', shrink=0.05))
ax.axis([0, 10, 0, 10])
plt.show()
```



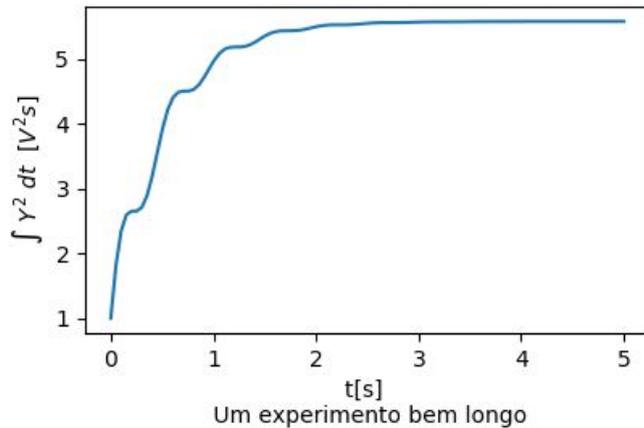
16.18 Escrevendo latex

```
import matplotlib.pyplot as plt
import numpy as np
x1 = np.linspace(0.0, 5.0, 100)
y1 = np.cos(2 * np.pi * x1) * np.exp(-x1)
fig, ax = plt.subplots(figsize=(5, 3))
fig.subplots_adjust(bottom=0.2, left=0.2)
```

```

ax.plot(x1, np.cumsum(y1**2))
ax.set_xlabel('t[s] \n Um experimento bem longo')
ax.set_ylabel(r'$\int Y^2 dt \ [V^2 s]$')
plt.show()

```

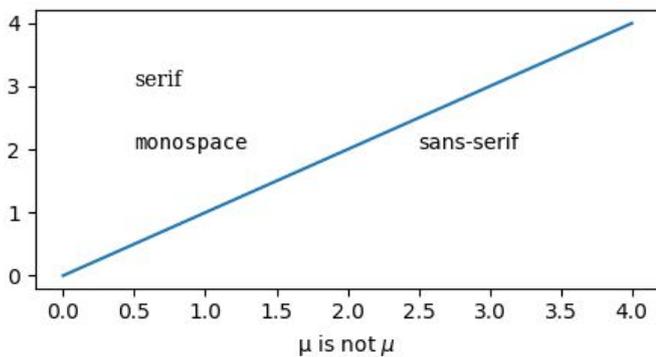


16.19 Mais texto

```

import matplotlib.pyplot as plt
plt.figure(figsize=(4.5, 2.5))
plt.plot(range(5))
plt.text(0.5, 3., "serif", family="serif")
plt.text(0.5, 2., "monospace", family="monospace")
plt.text(2.5, 2., "sans-serif", family="sans-serif")
plt.xlabel(u"$\mu$ is not $\mu$")
plt.tight_layout(.5)
plt.show()

```

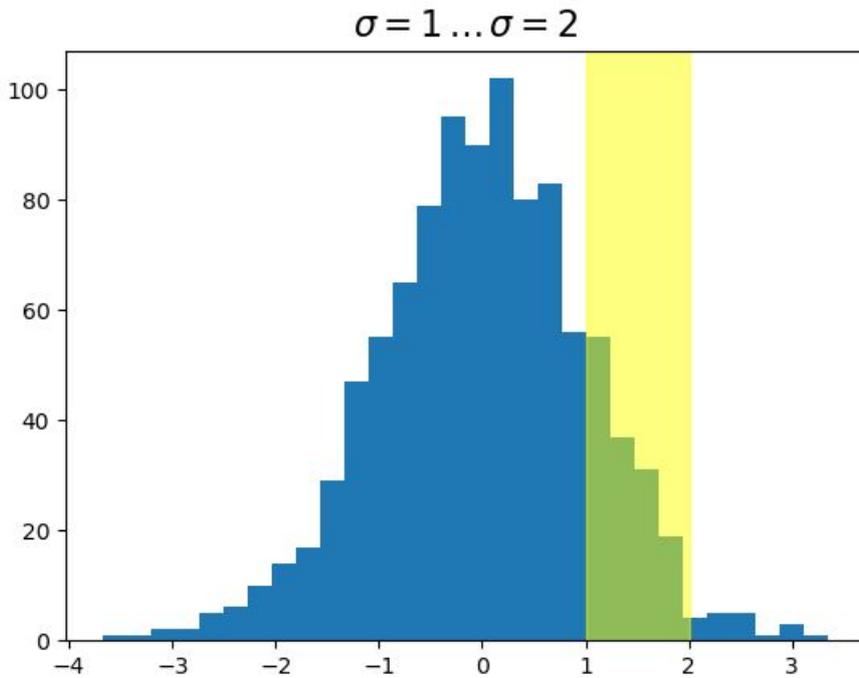


16.20 mais

```

import matplotlib.transforms as transforms
import matplotlib.patches as patches
fig = plt.figure()
ax = fig.add_subplot(111)
x = np.random.randn(1000)
ax.hist(x, 30)
ax.set_title(r'$\sigma=1 \ / \ \dots \ / \ \sigma=2$',
fontsize=16)
trans = transforms.blended_transform_factory(
ax.transData, ax.transAxes)
rect = patches.Rectangle((1, 0), width=1, height=1,
transform=trans, color='yellow',
alpha=0.5)
ax.add_patch(rect)
plt.show()

```



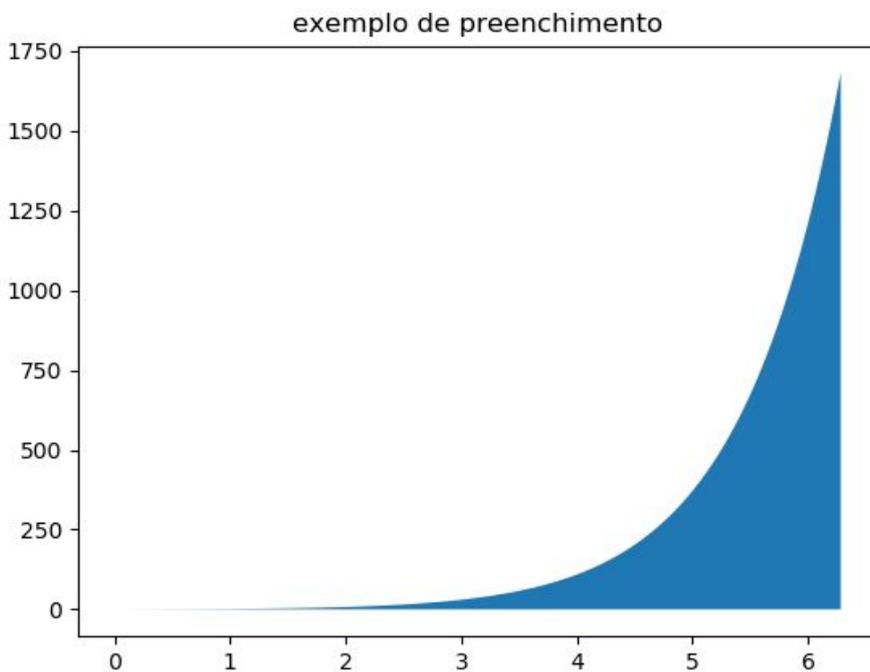
16.21 Preenchimento

Para fazer o preenchimento entre uma variável (aqui `ga` e o zero, faça:

```
axe.fill_between(range(12),ga,0)
```

como no exemplo a seguir:

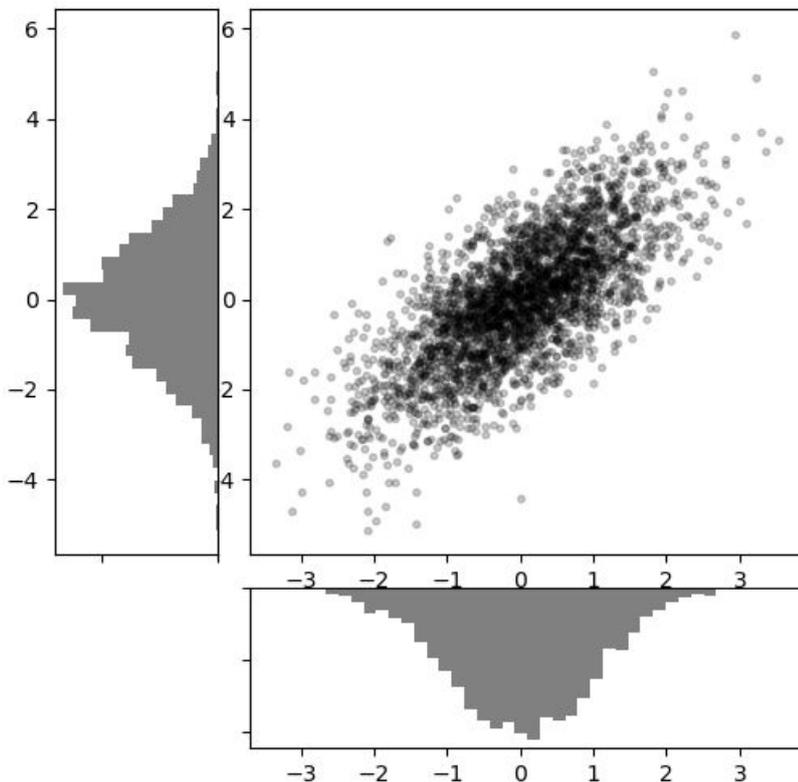
```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(0, 2*np.pi, 400)
y = np.exp(x)*(x/2)
yy=np.zeros((400))
fig, ax = plt.subplots()
plt.title('exemplo de preenchimento')
ax.fill_between(x,y,yy)
plt.show()
```



16.22 Exemplo com grid

```
import matplotlib.pyplot as plt
import numpy as np

# Create some normally distributed data
mean = [0, 0]
cov = [[1, 1], [1, 2]]
x, y = np.random.multivariate_normal(mean,
    cov, 3000).T
# Set up the axes with gridspec
fig = plt.figure(figsize=(6, 6))
grid=plt.GridSpec(4,4,hspace=0.2,wspace=0.2)
main_ax = fig.add_subplot(grid[:-1, 1:])
y_hist = fig.add_subplot(grid[:-1, 0],
    xticklabels=[], sharey=main_ax)
x_hist = fig.add_subplot(grid[-1, 1:],
    yticklabels=[], sharex=main_ax)
# scatter points on the main axes
main_ax.plot(x,y,'ok',markersize=3,alpha=0.2)
# histogram on the attached axes
x_hist.hist(x, 40, histtype='stepfilled',
    orientation='vertical', color='gray')
x_hist.invert_yaxis()
y_hist.hist(y, 40, histtype='stepfilled',
    orientation='horizontal', color='gray')
y_hist.invert_xaxis()
plt.show()
```



16.23 Espiral de Fermat

Em coordenadas polares:
 $r = \theta^{1/2}$ ou ainda $r^2 = a^2 \theta$

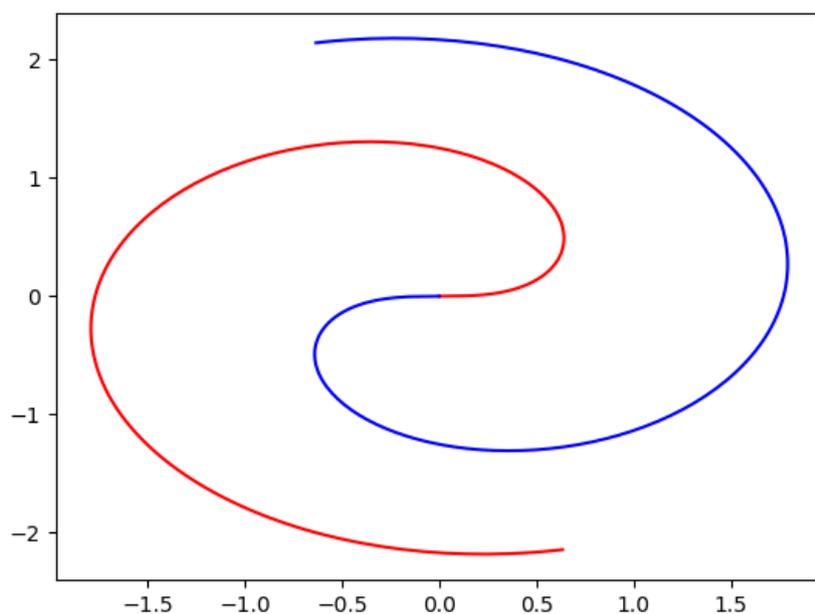
```
#fermaspi
import matplotlib.pyplot as plt
```

```

import numpy as np
def cart2pol(x, y):
    rho = np.sqrt(x**2 + y**2)
    phi = np.arctan2(y, x)
    return(rho, phi)

def pol2cart(rho, phi):
    x = rho * np.cos(phi)
    y = rho * np.sin(phi)
    return(x, y)
ta=200
t=np.linspace(0.0, 5, ta)
x=np.zeros((ta))
y=np.zeros((ta))
i=0
a=1
while(i<ta):
    (xa,ya)=pol2cart((a*t[i]**0.5),t[i])
    x[i]=xa
    y[i]=ya
    i=i+1
plt.plot(x,y,'r-')
i=0
a=-1
while(i<ta):
    (xa,ya)=pol2cart((a*t[i]**0.5),t[i])
    x[i]=xa
    y[i]=ya
    i=i+1
plt.plot(x,y,'b-')
plt.show()

```



16.24 Lemniscata

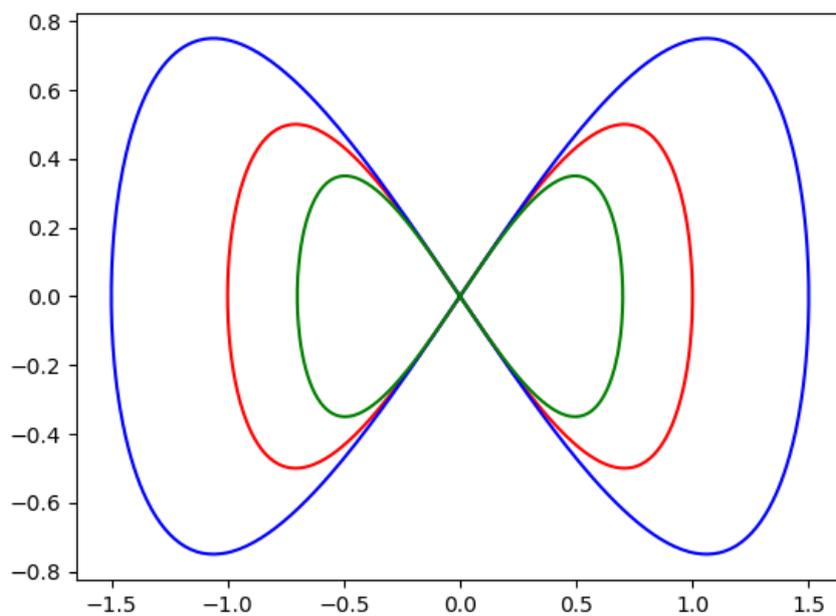
Em coordenadas cartesianas:

$$(x^2 + y^2)^2 = 2a^2(x^2 - y^2)$$

Em coordenadas polares:

$$r^2 = 2a^2 \cos 2\theta$$

```
#lemniscata
import matplotlib.pyplot as plt
import numpy as np
ta=500
t=np.linspace(0.0, 6.28, ta)
x=np.zeros((ta))
y=np.zeros((ta))
i=0
a=1
while(i<ta):
    x[i]=a*np.sin(t[i])
    y[i]=a*np.sin(t[i])*np.cos(t[i])
    i=i+1
plt.plot(x,y,'r-')
i=0
a=1.5
while(i<ta):
    x[i]=a*np.sin(t[i])
    y[i]=a*np.sin(t[i])*np.cos(t[i])
    i=i+1
plt.plot(x,y,'b-')
i=0
a=0.7
while(i<ta):
    x[i]=a*np.sin(t[i])
    y[i]=a*np.sin(t[i])*np.cos(t[i])
    i=i+1
plt.plot(x,y,'g-')
plt.show()
```



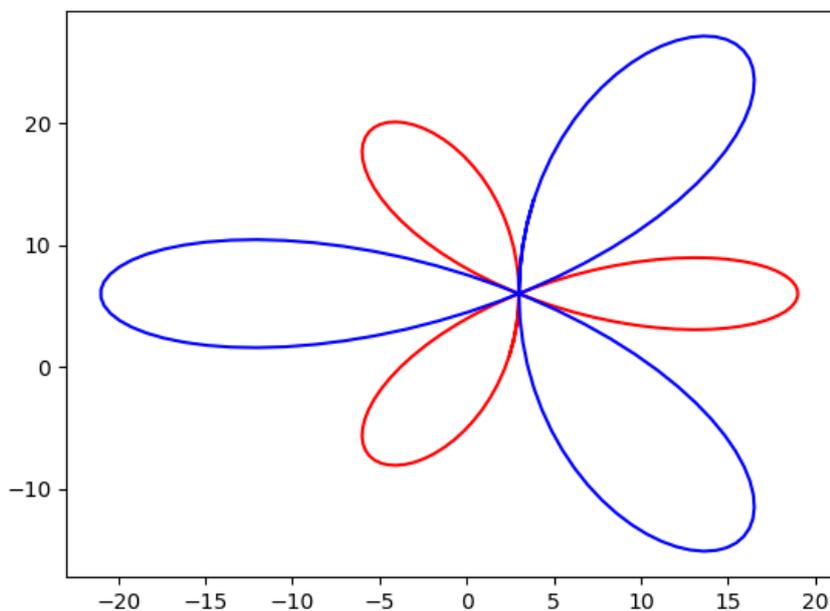
16.25 trifóide

```
#trifoide
import matplotlib.pyplot as plt
import numpy as np
fi=np.linspace(0.0, 6.5, 120)
x=np.zeros((120))
```

```

y=np.zeros((120))
i=0
a=-2
x0=3
y0=6
while(i<120):
    x[i]=x0+4*a*np.cos(fi[i])-4*a*np.cos(2*fi[i])
    y[i]=y0+4*a*np.sin(fi[i])+4*a*np.sin(2*fi[i])
    i=i+1
plt.plot(x,y,'r-')
i=0
a=3
x0=3
y0=6
while(i<120):
    x[i]=x0+4*a*np.cos(fi[i])-4*a*np.cos(2*fi[i])
    y[i]=y0+4*a*np.sin(fi[i])+4*a*np.sin(2*fi[i])
    i=i+1
plt.plot(x,y,'b-')
plt.show()

```



16.26 cardióide

Representação paramétrica:

$$\begin{aligned}
 x(\varphi) &= 2a(1 - \cos\varphi) \cdot \cos\varphi \\
 y(\varphi) &= 2a(1 - \cos\varphi) \cdot \sin\varphi \quad 0 \leq \varphi < 2\pi
 \end{aligned}$$

Em coordenadas polares:

$$r(\varphi) = 2a(1 - \cos\varphi)$$

Em coordenadas cartesianas:

$$(x^2 + y^2)^2 + 4ax(x^2 + y^2) - 4a^2y^2 = 0$$

#cardioide

```

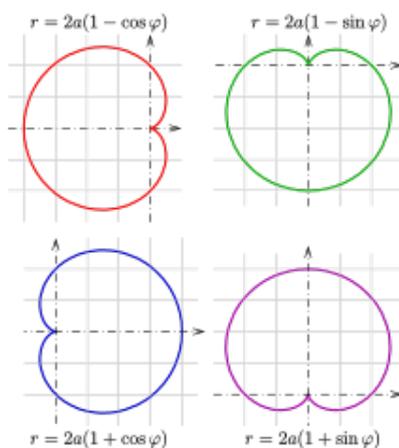
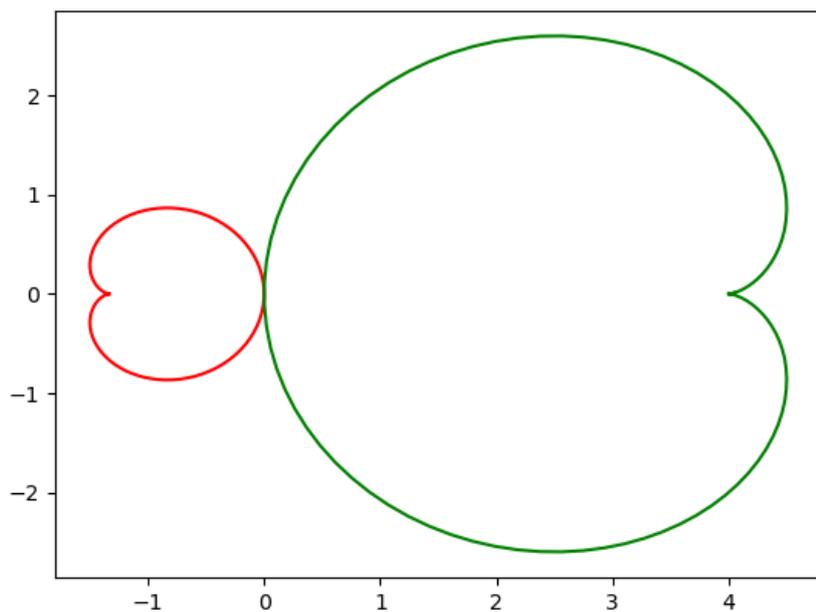
import matplotlib.pyplot as plt
import numpy as np
teta=np.linspace(0.0, 6.5, 120)
x=np.zeros((120))

```

```

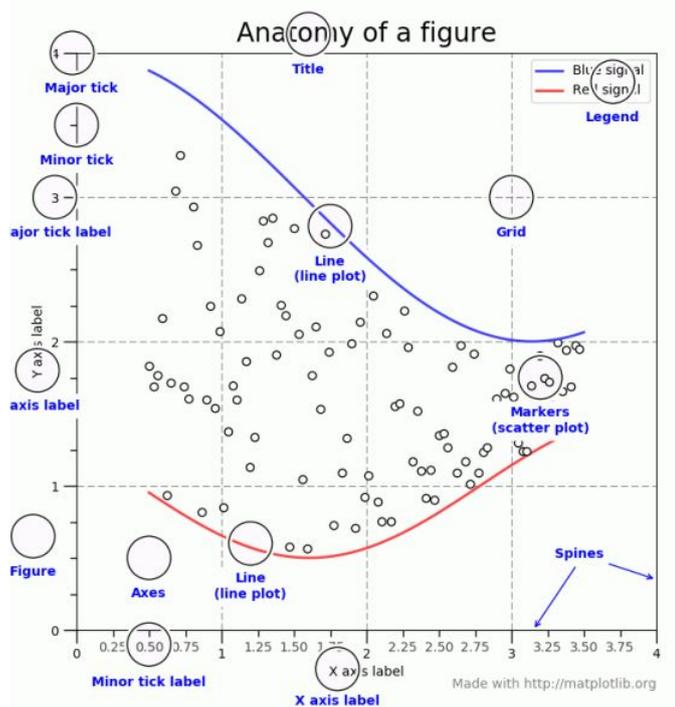
y=np.zeros((120))
i=0
a1=-2
while(i<120):
    x[i]=((2/3)*a1)+((1/3)*a1)*np.cos(teta[i])*(1-np.cos(teta[i]))
    y[i]=((1/3)*a1)*np.sin(teta[i])*(1-np.cos(teta[i]))
    i=i+1
plt.plot(x,y,'r-')
a2=6
i=0
while(i<120):
    x[i]=((2/3)*a2)+((1/3)*a2)*np.cos(teta[i])*(1-np.cos(teta[i]))
    y[i]=((1/3)*a2)*np.sin(teta[i])*(1-np.cos(teta[i]))
    i=i+1
plt.plot(x,y,'g-')
plt.show()

```



Para desenhar a cardioide em outras posições, use:

16.27 Anatomia de uma figura



Capítulo 17

Pacote: OpenCV

Definição

O pacote OCV (*Open Source Computer Vision*) agrega funções para processamento e análise de imagens. O pacote foi desenvolvido originalmente em C++ e rapidamente migrado para Python. Usa como pré-requisitos os seguintes pacotes

- Numpy: numerical python
- Matplotlib: gerador de gráficos no estilo Matlab

Não precisa se preocupar, se o instalador detectar que estes pacotes não estão presentes ele (o instalador) vai carregar e instalar estes dois pacotes.

A instalação do openCV se dá pelo comando

```
pip install opencv-python
```

emitido no diretório onde estiver instalado o pip.exe. Tipicamente é

```
cd\python\scripts
```

Depois de instalar o pacote, teste o funcionamento do mesmo dentro do python emitindo:

```
>>>import cv2
```

Se o python responder >>> sem nenhuma outra mensagem, é porque a instalação deu certo.

Algumas funções

A seguir uma lista de algumas funções do pacote

Função	o que faz	forma de chamar
Ler uma imagem	carrega para dentro do programa uma ou mais matrizes (vide canal)	<code>im=cv2.imread('arquivo.jpg',dica)</code>

`dica` é opcional e indica como a leitura deve ser feita. Seus valores:

1 leia o arquivo em true color

0 leia o arquivo em níveis de cinza

-1 leia o arquivo como ele foi criado

Após este comando, a variável `im` tem associada as seguintes informações:

variável	conteúdo
<code>im.shape[0]</code>	altura da imagem em pixels
<code>im.shape[1]</code>	largura da imagem em pixels
<code>im.shape[2]</code>	quantidade de canais da imagem

Função	o que faz	forma de chamar
Mostrar uma imagem	abre uma janela e escreve nela a imagem	<code>im=cv2.imshow('nome da janela',im)</code>
Gravar a imagem em disco	grava a imagem (eventualmente alterada)	<code>cv2.imwrite("arquivo.jpg",im)</code>

Se a imagem for branco e preto, haverá um único canal (uma matriz com o nome de `im`). Se a imagem for true color, haverá 3 matrizes de mesmo tamanho, a primeira sendo `blue` a segunda `green` e a terceira `red`. Acompanhe o programa abaixo

```
def ima1():
    import cv2
    imagem=cv2.imread('jovem.jpg')
    for i in range(0,imagem.shape[0],10):
        for j in range(0,imagem.shape[1],10):
            # (a,b,c)=imagem[i,j]
            imagem[i:i+5,j:j+5]=(0,255,255)
    cv2.imshow('olhe',imagem)
ima1()
```

Função	o que faz	forma de chamar
Desenha uma linha	desenha uma linha na imagem	cv2.line(im, (origem), (destino), cor)
Desenha um retângulo vazado	cria um retângulo se largura = -1, figura é cheia. Default é 1.	cv2.rectangle(im,(esqsup),(dirinf0, cor, largura-risco)
Desenha circulo	desenha um circulo	cv2.circle(im, (centro), raio, cor)

Acompanhe o programa

```
import numpy as np
import cv2
im = cv2.imread('oba.jpg')
vermelho = (0,0,255)
verde=(0,255,0)
azul=(255,0,0)
cv2.line(im,(5,5),(30,60), verde)
cv2.rectangle(im,(20,30),(100,120),azul,-1)
(X, Y) = (im.shape[1]//2, im.shape[0]//2)
cv2.circle(im,(X,Y),30,azul)
```

Para recortar (crop) uma imagem, faça:

```
import cv2
im=cv2.imread('oba.jpg')
recorte = im[100:200, 100:200]
cv2.imshow("Recorte",recorte)
cv2.imwrite("recorte.jpg",recorte)
```

Função	o que faz	forma de chamar
redimensionar	redimensiona uma imagem	imn=cv2.resize(im,(tamanho novo), interpolation=método)

o método

pode ser:

cv2.INTER_AREA for shrinking

cv2.INTER_CUBIC (slow)

cv2.INTER_LINEAR for zooming.

Por default o método usado é **INTER_LINEAR**. Veja o código

```
import numpy as np
import cv2
im=cv2.imread('oba.jpg')
largura=im.shape[1]
altura=im.shape[0]
proporcao=float(altura/largura)
largura_nova=500
altura_nova=int(largura_nova*proporcao)
tamanho_novo=(largura_nova, altura_nova)
imredimens=cv2.resize(im,tamanho_novo,interpolation=cv2.INTER_AREA)
cv2.imshow('Resultado',imredimens)
cv2.waitKey(0)
```

A seguir, como fazer flip vertical e horizontal

```
import cv2
im = cv2.imread('oba.jpg')
flip_hor = im[::-1,:]
flip_vert = im[:,::-1]
flip_ambos = im[::-1,::-1]
... gravar, mostrar, etc
```

Para rotar uma imagem:

```
def ri(): #roda imagem
import cv2
im=cv2.imread("pedro_jovem.jpg")
(alt,lar)=im.shape[:2]
centro=(lar//2,alt//2)
m=cv2.getRotationMatrix2D(centro,-10,1.0) #-10 graus
imr=cv2.warpAffine(im,m,(lar,alt))
cv2.imshow("rodada",imr)
cv2.waitKey(0)
ri()
```


Capítulo 18

Pacote: TKInter

Este pacote permite criar software Python com interface GUI (cara de aplicativo windows).

Agora vamos estender um pouco a estratégia de desenvolvimento do TK. Começando do começo. Lembrando que o TK está presente por default nas instalações de python. Para verificar, faça

```
>>> import tkinter
>>> tkinter._test()
```

Deve aparecer um caixinha com um botão, para você clicar nele. Outra verificação que pode ser feita é

```
ttkinter.Tcl (). eval ('info patchlevel')
```

que responde com o nível do pacote (aqui deu '8.6.8').

18.1 Hello World em Tkinter

Um primeiro programa poderia ser

```
from tkinter import *
from tkinter import ttk
root = Tk()
ttk.Button(root, text="Hello World").grid()
root.mainloop()
```

Salvando este programa em algum diretório, com o nome de `hello.py` depois deve-se fazer

```
python hello.py
```

Deve desenhar uma caixinha simples.

18.2 Programa Inicial

Eis aqui uma aplicação simples

```
from tkinter import *
from tkinter import ttk

def funcao1():
    def funcao2(*args):
        try:
            v=float(c1.get())
            c2.set(v**2*3.141592)
        except ValueError:
            pass

root=Tk()
root.title("Calcula área de circulo")
mf=ttk.Frame(root)
mf.grid()
c1=StringVar()
c2=StringVar()
```

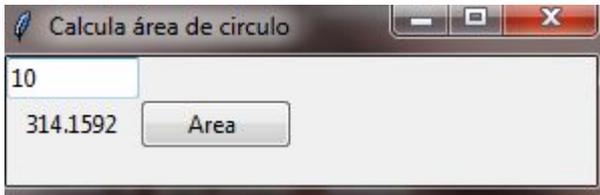
```

fe=ttk.Entry(mf,width=10,textvariable=c1)
fe.grid(column=0,row=0)
ttk.Button(mf,text="Area ",command=funcao2).grid(column=1,row=1)
ttk.Label(mf,textvariable=c2).grid(column=0,row=1)
fe.focus()
root.bind('<Return>',funcao2)
root.mainloop()

```

funcao1()

Eis o resultado



Muito importante, a lista de coisas que não precisamos fazer (o Tk faz)

- nós não temos que nos preocupar em redesenhar a tela quando as coisas mudaram
- não precisamos nos preocupar com a análise e o envio de eventos, a detecção de ocorrências ou o tratamento de eventos em cada widget
- nós não tivemos que fornecer muitas opções quando criamos widgets; os padrões pareciam tomar conta da maioria das coisas, e então nós só tínhamos que mudar coisas como o texto que o botão mostrava
- não precisávamos escrever código complexo para obter e definir os valores de widgets simples; nós apenas os anexamos às variáveis
- nós não precisamos nos preocupar com o que acontece quando o usuário fecha a janela ou redimensiona
- nós não precisamos escrever código extra para fazer tudo isso funcionar em multiplataforma

18.3 Gerenciamento da geometria

Para ter os widgets realmente colocados na janela da tela e sobretudo onde nas janelas eles aparecem é uma etapa separada chamada de gerenciamento da geometria.

No exemplo acima quem faz isso é o comando **grid** para o qual passamos qual a linha e a coluna onde queremos inserir o widget. Há outros gerenciadores de geometria em Tk. Outro é o **pack**. O trabalho de qualquer gerente de geometria é descobrir exatamente onde desenhar cada coisa. Para fazer isso o gerenciador deve equilibrar várias restrições, como por exemplo

- Cada widget pode ter um tamanho natural (talvez determinado pelo texto e pela fonte), mas o nível superior de todos os widgets não é grande o suficiente para acomodá-los.
- Se a janela é maior do que todos os widgets, como o tamanho extra será usado ?
- Se a janela é redimensionada, como o novo desenho é reconstruído ?
- Como itens de tamanhos diferentes devem ser alinhados para manter a estética da janela ?
- Não esquecer que em uma interface complexa, pode haver quadros aninhados dentro de outros quadros e todos eles precisam ser considerados.

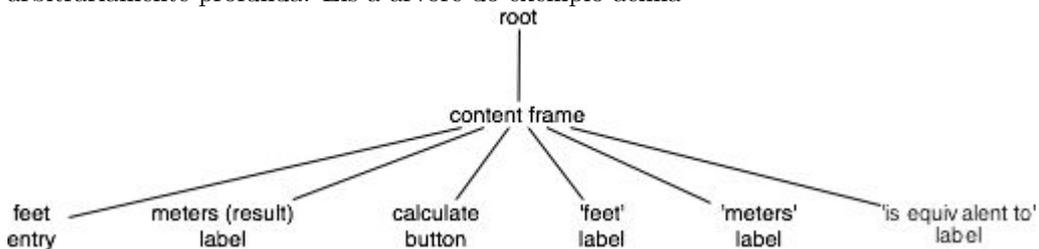
O gerenciador depende do conceito de widget mestre e widget escravo. Um mestre é uma janela ou um quadro que conterá outros widgets que são chamados de escravos

O gerenciador perguntará a cada escravo quão grande ele deveria ser para ser exibido corretamente. Em seguida ele aplica quaisquer medida adicional que o programa resolva aplicar neste escravo em particular. No exemplo acima passou-se a linha e a coluna de cada widget e também o parâmetro **sticky** (pegajoso) para sugerir como o widget deveria ser alinhado, possivelmente esticado. Já os parâmetros **columnconfigure** e **rowconfigure** indicam linhas e colunas que deveriam expandir se houver espaço extra disponível na janela.

Depois de obter todos os dados dos escravos o gerenciador negocia com o widget mestre a fim de construir o quadro a mostrar.

18.4 Widgets

São as coisas que você vê na tela: um botão, uma entrada, um quadro, alguns rótulos. Também caixas de seleção, visualizações de árvores, barras de rolagem, etc etc. Um widget é um objeto instanciado de classes que representam todos esses controles. Todos os widgets são criados dentro de uma hierarquia, que começa numa raiz. Ela pode ser arbitrariamente profunda. Eis a árvore do exemplo acima



Ao criar um widget você deve passar o pai como parâmetro. A única exceção é a janela raiz que é a janela superior da hierarquia, que conterá todo o resto. Veja o exemplo

```
root = Tk()
content = ttk.Frame(root)
button = ttk.Button(content)
```


Capítulo 19

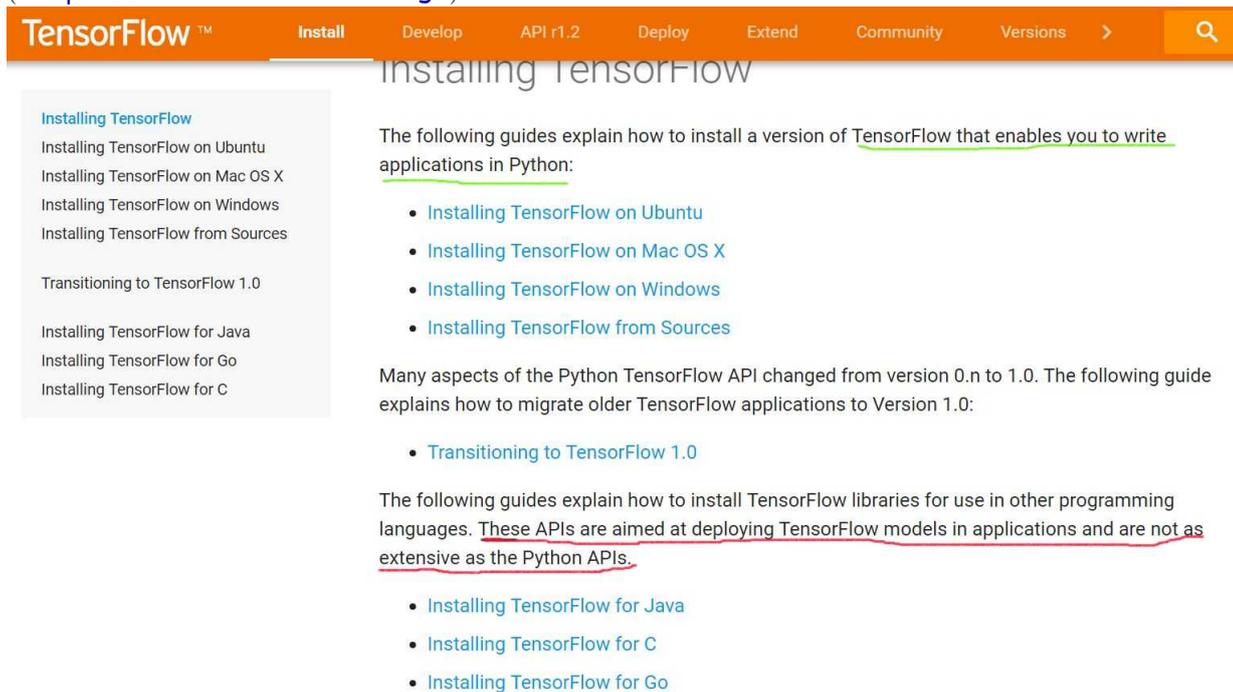
Pacote: TensorFlow

É um produto freeware (licença Apache) desenvolvido pela Google para aprendizado de máquina. Substituiu seu antecessor DistBelief que era proprietário. Além de ser um belo projeto de pesquisa ele é usado em produção em muitos produtos google (tradução, classificação de fotos, busca, reconhecimento da fala humana, etc). Sua primeira versão foi liberada em 9/11/15 e seu desenvolvimento começou em 2011.

Está em www.tensorflow.org. Embora tenha APIs para uso em C++, Java, Go seu uso mais completo e documentado se dá em Python, já que a ferramenta está desenvolvida neste ambiente. Uma ótima oportunidade para aprender e começar a usar esta linguagem.

Pode-se visualizar o vídeo em https://www.youtube.com/watch?v=oZikw5k_2FM. Se quiser, peça a tradução simultânea para o português. Verá o TS em ação.

Eis aqui uma demonstração do vigor do Python neste ambiente. É uma das telas do TensorFlow, lá no site dele (<https://www.tensorflow.org/>)



The screenshot shows the TensorFlow website's 'Installing TensorFlow' page. The navigation bar includes links for 'Install', 'Develop', 'API r1.2', 'Deploy', 'Extend', 'Community', and 'Versions'. The main content area is titled 'Installing TensorFlow' and contains the following text:

The following guides explain how to install a version of TensorFlow that enables you to write applications in Python:

- [Installing TensorFlow on Ubuntu](#)
- [Installing TensorFlow on Mac OS X](#)
- [Installing TensorFlow on Windows](#)
- [Installing TensorFlow from Sources](#)

Many aspects of the Python TensorFlow API changed from version 0.n to 1.0. The following guide explains how to migrate older TensorFlow applications to Version 1.0:

- [Transitioning to TensorFlow 1.0](#)

The following guides explain how to install TensorFlow libraries for use in other programming languages. These APIs are aimed at deploying TensorFlow models in applications and are not as extensive as the Python APIs.

- [Installing TensorFlow for Java](#)
- [Installing TensorFlow for C](#)
- [Installing TensorFlow for Go](#)

Vale lembrar a existência de um ambiente Python que pode ser executado avulso (a partir de um pendrive, por exemplo), o que é ótimo quando você não tem os direitos de administrador na máquina que está usando. Trata-se do chamado Python portátil e ele pode ser buscado em <https://winpython.github.io/>. Este sujeito já traz o NUMPY instalado.

Tensor Flow

Este produto só roda em máquinas 64 bits (lembrem-se, ele não economiza recursos) e estando nesse ambiente, a instalação é simples, basta fazer `pip install tensorflow`.

Depois de instalar, para verificar se deu tudo certo faça:

```
>>> import tensorflow as tf
>>> hello = tf.constant('Oi, TensorFlow!')
>>> sess = tf.Session()
>>> print(sess.run(hello))
```



```

from tensorflow.examples.tutorials.mnist import input_data
import tensorflow as tf #importa o tensorflow

FLAGS = None

def main(_):
    # Import data
    mnist = input_data.read_data_sets(FLAGS.data_dir, one_hot=True)
    # Create the model
    x = tf.placeholder(tf.float32, [None, 784]) # dados de entrada
    W = tf.Variable(tf.zeros([784, 10]))      # a RN
    b = tf.Variable(tf.zeros([10]))          # os bias
    y = tf.matmul(x, W) + b                  # o resultado da RN
    # Define loss and optimizer
    y_ = tf.placeholder(tf.float32, [None, 10])
        # lugar onde vem a resposta certa
    cross_entropy = tf.reduce_mean(
        tf.nn.softmax_cross_entropy_with_logits(labels=y_, logits=y))
        # quero minimizar a diferença entre y (calculado) e y_ (certo)
    train_step =
    tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)
    sess = tf.InteractiveSession() # cria a sessao
    tf.global_variables_initializer().run()
        # inicializa variáveis
    # Train
    for _ in range(1000):
        # treinamento 1000 sessoes de 100 aleatorios
        batch_xs, batch_ys = mnist.train.next_batch(100)
        # coloca aqui os dados
        sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})
        #roda a RN
    # Test trained model
    correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
        #y é igual a y_?
    accuracy =
    tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
        #acurácia
    print(sess.run(accuracy, feed_dict={x: mnist.test.images,
        y_: mnist.test.labels}))
        #imprime acurácia a partir das imagens de teste...
if __name__ == '__main__':
    # descobre se isto está sendo chamado ou importado
    parser = argparse.ArgumentParser()
    parser.add_argument('--data_dir',
        type=str, default='/tmp/tensorflow/mnist/input_data',
        help='Directory for storing input data')
    FLAGS, unparsed = parser.parse_known_args()
    tf.app.run(main=main, argv=[sys.argv[0]] + unparsed)

```

Se você rodar o programa acima terá uma rede neural funcional completa e obterá a acurácia de uma execução (tipicamente 92%).

Um exercício

Sua tarefa aqui é razoavelmente simples:

1. Descubra o mínimo necessário para entender uma rede neural artificial
2. Instale (ou descubra o uso de) Python 3 em uma máquina 64 bits
3. Instale o pacote tensorflow (`pip install tensorflow`)
4. Estude os tutoriais:
 - Getting started

-
- ... with Tensor Flow
 - MNIST for ML beginners
 - Deep MNIST for Experts

Se quiser, peça para o browser traduzir para o português, o resultado é bastante razoável (feito pelo TF...)

5. Digite ou copie o programa `mnist_softmax.py`

6. Rode-o.

Você terá uma Rede Neural satisfatória e contendo o estado da arte em Machine Learning.

Capítulo 20

Pacote: Itertools

O módulo itertools implementa iteradores inspirado em APL, Haskell e SML. Cada um deles foi reconstruído para se adaptar a Python eficiente.

Iteradores infinitos Precisam ser interrompidos na marra

Iterador	Argumentos	Resultados	Exemplo
count()	start, [step]	start, start+step, start+2*step, count(10) -> 10 11 12 13 14 ...
cycle()	p	p0, p1, ... plast, p0, p1, ...	cycle('ABCD') -> A B C D A B C D ...
repeat()	elem [,n]	elem, elem, elem, ... endlessly or up to n times	repeat(10, 3) -> 10 10 10

A seguir, iteradores que terminam quando acaba a lista mais curta:

Iterador	Argumentos	Resultados	Exemplo
accumulate()	p [,func]	p0, p0+p1, p0+p1+p2, ...	accumulate([1,2,3,4,5]) -> 1 3 6 10 15
chain()	p, q, ...	p0, p1, ... plast, q0, q1, ...	chain('ABC', 'DEF') -> A B C D E F
chain.from_iterable()	iterable	p0, p1, ... plast, q0, q1, ...	chain.from_iterable(['ABC', 'DEF']) -> A B C D E F
compress()	data, selectors	(d[0] if s[0]), (d[1] if s[1]), ...	compress('ABCDEF', [1,0,1,0,1,1]) -> A C E F
filterfalse()	pred, seq	elements of seq where pred(elem) is false	filterfalse(lambda x: x%2, range(10)) -> 0 2 4 6 8
groupby()	iterable[, key]	sub-iterators grouped by value of key(v)	
islice()	seq, [start,] stop [, step]	elements from seq[start:stop:step]	islice('ABCDEFGF', 2, None) -> C D E F G
starmap()	func, seq	func(*seq[0]), func(*seq[1]), ...	starmap(pow, [(2,5), (3,2), (10,3)]) -> 32 9 1000
takewhile()	pred, seq	seq[0], seq[1], until pred fails	takewhile(lambda x: x<5, [1,4,6,4,1]) -> 1 4
tee()	it, n	it1, it2, ... itn splits one iterator into n	
zip_longest()	p, q, ...	(p[0], q[0]), (p[1], q[1]), ...	zip_longest('ABCD', 'xy', fillvalue='-') -> Ax By C-D-

geradores combinatórios:

Iterador	Argumentos	Resultados
product()	p, q, ... [repeat=1]	cartesian product, equivalent to a nested for-loop
permutations()	p[, r]	r-length tuples, all possible orderings, no repeated elements
combinations()	p, r	r-length tuples, in sorted order, no repeated elements
combinations_with_replacement()	p, r	r-length tuples, in sorted order, with repeated elements
product('ABCD', repeat=2)		AA AB AC AD BA BB BC BD CA CB CC CD DA DB DC DD
permutations('ABCD', 2)		AB AC AD BA BC BD CA CB CD DA DB DC
combinations('ABCD', 2)		AB AC AD BC BD CD
combinations_with_replacement('ABCD', 2)		AA AB AC AD BB BC BD CC CD DD

20.1 Funções do Pacote Itertools

As funções a seguir retornam iteradores. Algumas devolvem fluxos de tamanho infinito, portanto este deve ser interrompido em algum momento.

itertools.chain(*iterables) Retorna cada um dos elementos iteráveis. Veja

```
import itertools as it
def exe1():
    for a in it.chain('ABCDEF'):
        print(a)
exe1()
```

Este caso é exatamente igual a

```
import itertools as it
def exe1():
    for a in list('ABCDEF'):
        print(a)
exe1()
dá como resposta A, B, C, D, E e F
```

itertools.combinations(iterable, r) Devolve as combinações r a r . Devolve subsequências de comprimento r a partir da entrada. As combinações são emitidas em ordem lexicográfica. Se a entrada já está ordenada, a saída também estará ordenada. Os elementos são tratados unicamente por sua posição, não por seu valor. Se os elementos na entrada são únicos, não haverá repetição na saída.

O número de itens retornados é

$$C(n, r) = \frac{n!}{r!(n-r)!} = \frac{A(n, r)}{r!}$$

Veja o exemplo

```
import itertools as it
def exe1():
    for a in it.combinations('ABCDEF',3):
        print(a)
exe1()
```

Dá como respostas

```
('A', 'B', 'C')
('A', 'B', 'D')
('A', 'B', 'E')
('A', 'B', 'F')
('A', 'C', 'D')
('A', 'C', 'E')
('A', 'C', 'F')
('A', 'D', 'E')
```

```
...
('D', 'E', 'F')
```

Ou

```
it.combinations(range(5),3)
dá (0, 1, 2), (0, 1, 3), (0, 1, 4), (0, 2, 3),
(0, 2, 4), (0, 3, 4), (1, 2, 3), (1, 2, 4),
(1, 3, 4) e (2, 3, 4)
```

itertools.combinations_with_replacement(iterable, r) Retorna subsequências de comprimento r a partir da entrada, permitindo que os elementos da entrada sejam repetidos mais de uma vez.

O número de itens retornados é

$$(n + r - 1)!/r!(n - 1)!$$

quando $n > 0$.

veja o exemplo:

```
it.combinations_with_replacement('ABC',2)
dá
('A', 'A'), ('A', 'B'), ('A', 'C'),
('B', 'B'), ('B', 'C') e ('C', 'C')
```

itertools.compress(data, selectors) Retorna apenas os iteradores que são selecionados como True. Veja:

```
it.compress('ABCDE',[1,0,1,0,1]):
Retorna A, C e E
```

itertools.count(start=0, step=1) Retorna valores espaçados em $step$ começando em $start$.

itertools.cycle(iterable) Retorna os itens da entrada de maneira cíclica ao infinito. Precisa ser interrompido.

itertools.dropwhile(predicate, iterable) Enquanto o predicado for falso, ele não devolve. Após o primeiro Verdadeiro, volta tudo.

```
it.dropwhile(lambda x: x.isupper(),'AcBgChDE'):
é cBgChDE
```

itertools.groupby(iterable[, key]) Devolve duas respostas, veja:

```
import itertools as it
data='aaaaaaabbbbbbbccccccde'
groups = []
uniquekeys = []
for k, g in it.groupby(data):
    groups.append(list(g))      # Armazena como uma lista
    uniquekeys.append(k)
print(groups)
print(uniquekeys)
```

Retorna

```
print(k)
[['a', 'a', 'a', 'a', 'a', 'a', 'a'], ['b', 'b', 'b', 'b', 'b', 'b', 'b', 'b'], ['c', 'c', 'c', 'c', 'c', 'c'], ['d'], ['e']]
```

```
print(g)
['a', 'b', 'c', 'd', 'e']
```

filterfalse Substitui as `ifilter` e `ifilterfalse` do Python2 que não existem mais no 3.

```
import itertools
def is_lt_100(x):
    return x < 100

a = [1,2,3,4,1000,66,1201]
b = itertools.filterfalse(is_lt_100, a)
```

```
for i in b:
    print (i)
```

deu 1000 e 1201

map A rigor este cara não é do itertools, mas vamos lá

```
for a in map(pow,(2,3,5),(10,11,12)):
    print(a)
deu como respostas 1024, 177147 e 244140625
```

itertools.starmap(function, *iterables) Compare com a anterior

```
import itertools as it
for a in it.starmap(pow,[(2,10),(3,11),(5,12)]):
    print(a)
```

itertools.permutations(iterable, r=None)

```
\begin{verbatim}
import itertools as it
for a in it.permutations('ABC',2):
    print(a)

deu
('A', 'B'), ('A', 'C'), ('B', 'A'),
('B', 'C'), ('C', 'A') e ('C', 'B')
```

itertools.product(iterables, repeat=1) Realiza o produto cartesiano. O `product(A,B)` faz o mesmo que `((x,y) for x in A for y in B)`.

Para comutar o produto de um iterável com ele mesmo use a palavra `repeat`. Assim, `product(A,repeat=4)` faz o mesmo que `product(A,A,A,A)`.

```
import itertools as it
for a in it.product('ABC','123'):
    print(a)

deu
('A', '1'), ('A', '2'), ('A', '3'),
('B', '1'), ('B', '2'), ('B', '3'),
('C', '1'), ('C', '2') e ('C', '3')
```

itertools.repeat(objeto[,vezes]) Repete indefinidamente um iterador. Veja em

```
>>> list(map(pow, range(10), repeat(2)))
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

itertools.takewhile(predicate, iterable) Retorna até que o predicado seja falso

```
# takewhile(lambda x: x<5, [1,4,6,4,1]) --> 1 4
```

Capítulo 21

UTF-8

A informática convive – desde seu início – com uma dificuldade e tanto: como padronizar a utilização de códigos binários. O objetivo é nobre: permitir a troca de objetos binários e a sua correta interpretação por parte de todos os parceiros em uma comunicação qualquer. A necessidade é nova, a informática só apareceu na face da terra na década de 50 do século passado. No início, cada fabricante usava seus códigos. A IBM, por exemplo, construiu toda a sua família de equipamentos usando o código EBCDIC. Isto evoluiu para o código ASCII (início de 1960, comite ANSI-X3.2) que acabou sendo adotado como padrão. Ele privilegiou o idioma inglês, e padronizou seus caracteres nas primeiras 128 posições do código ASCII. Aos demais idiomas do planeta ficou disponível a segunda parte do código ASCII (mais 128 posições). É aqui que o código BRASCII (NBR-9614:1986 e NBR-9611:1991 da ABNT) colocou os caracteres acentuados do português. Assim como o Brasil, muitos outros países fizeram o mesmo. Isto apresentou pelo menos 2 problemas:

- Um arquivo composto no Brasil, quando impresso numa impressora fabricada na França, aparecia como uma série de caracteres malucos.
- Idiomas com muitos caracteres (katakana, hiragana, hangul, tagalog,...) simplesmente não tinham como ser representados no código ASCII. Não havia espaço.

21.1 Unicode

Em meados da década de 80, a academia e a indústria começaram a discutir propostas de resolver os 2 problemas acima. Diversos padrões foram sugeridos e eles começaram a ser usados, na base do “cada um por si”. Uma primeira proposta sugeria usar 16 bits (2 bytes) o que permitiria $2^{16} = 65536$ caracteres. Parecia muito, mas vale citar Peter Norton, que disse há mais de 30 anos: *na informática, não importa quanto você tem. Não é o suficiente.*

Em 1991, criou-se o Consórcio Unicode e em outubro de 1991 o primeiro volume do padrão foi publicado. Em 1996, introduziu-se um mecanismo de codificação e com ele, a barreira dos 2 bytes foi ultrapassada. Agora, o espaço de endereçamento do Unicode ultrapassou o milhão de caracteres, o que permitiu incluir idiomas históricos (hieróglifos egípcios) ou mesmo idiomas muito pouco usados (Linear B por exemplo), além de algumas brincadeiras como o idioma Klingon. O padrão atual consagra 1.114.112 códigos de ponto (*codepoints*), correspondendo ao intervalo hexadecimal de 00.00.00 a 10.FF.FF. Note que nem todo codepoint indica um caracter, já que alguns deles são reservados a outras finalidades.

Há alguns codepoints privados, que não têm significação no Unicode. Eles ficam para uma negociação entre remetente e destinatário e podem ser usados para qualquer coisa (são eles: área privada: U+E000..U+F8FF com 6,400 caracteres; área suplementar A: U+F0000..U+FFFFD com 65,534 caracteres e área suplementar B: U+100000..U+10FFFF com 65,534 caracteres).

O Consórcio Unicode trabalha junto com a ISO (International Organization for Standardization) já que o padrão ISO/IEC 10646 é o próprio código Unicode. Um conceito importante é o de script. Trata-se de um conjunto de letras, sinais e regras de escrita que agregam partes do Unicode e que são usados em conjunto de *writing systems*. Na versão 7 do Unicode, são 123 os scripts, e há uma lista de candidatos a serem incluídos.

21.2 UTF=Unicode Transformation Formats

Diversos mecanismos foram sugeridos para implementar o código Unicode, com a preocupação sempre presente de compatibilidade reversa, isto é: um novo esquema de codificação deveria preservar – tanto quanto possível – o legado infundável de dados já coletados e guardados. Foram propostos os UTF-1 (nunca chegou a ser muito usado, pois tinha problemas de performance), UTF-16 (usava 1 ou 2 palavras de 16 bits), UTF-32 (cada caracter usava exatos 32 bits), mas todas elas acabaram migrando para a UTF-8 que se firmou como padrão de fato, principalmente porque muito cedo foi adotado como padrão pelo consórcio ICM (Internet Mail Consortium) e pelo W3C. O Google reporta que desde 2008, a maioria dos conteúdos HTML trocados na web usa a codificação UTF-8. O esquema todo foi escrito durante um jantar em 2 de setembro de 1992 por Ken Thompson e Rob Pike. Suas grandes qualidades são:

- Compatibilidade reversa, já que caracteres da primeira parte do ASCII não sofrem modificação (1º bit=0). Desta maneira qualquer arquivo ASCII que não use a segunda parte já é um arquivo UTF-8.
- Clara distinção entre caracteres de byte único (1º bit=0) e caracteres multi-bytes. Estes contém um primeiro byte heading e um ou mais bytes de continuação. O heading tem 2 ou mais bits 1 seguidos por um zero enquanto os bytes de continuação sempre começam por 10.
- Auto-sincronização. Bytes únicos, header e continuação não compartilham valores, indicando que o início do caracteres está a no máximo 3 bytes para trás.
- Indicação do comprimento: O número de uns na alta ordem do byte heading indica a quantidade de bytes do caractere, sem ser necessário examinar os seus bytes.

Veja-se um resumo do exposto

bits	início	fim	tamanho	byte1	byte2	byte3	byte4
7	U+0000	U+007F	1	0xxxxxxx			
11	U+0080	U+07FF	2	110xxxxx	10xxxxxx		
16	U+0800	U+FFFF	3	1110xxxx	10xxxxxx	10xxxxxx	
21	U+10000	U+1FFFFF	4	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx

Veja-se agora alguns exemplos de caracteres unicode de 1, 2, 3 e 4 bytes

Character	Binary code point
\$	U+0024 0100100
¢	U+00A2 000 10100010
€	U+20AC 00100000 10101100
報	U+24B62 00010 01001011 01100010

Binary UTF-8	Hexadecimal UTF-8
00100100	24
11000010 10100010	C2 A2
11100010 10000010 10101100	E2 82 AC
11110000 10100100 10101101 10100010	F0 A4 AD A2

Uma tabela de conversão binário hexadecimal para ajudar:

hex	bin	hex	bin	hex	bin	hex	bin
0	0000	4	0100	8	1000	C	1100
1	0001	5	0101	9	1001	D	1101
2	0010	6	0110	A	1010	E	1110
3	0011	7	0111	B	1011	F	1111

21.3 Exemplos

1. Seja o seguinte conjunto de 5 caracteres UNICODE codificados em UTF-8.

F2B1B88DF4BAA8A7E192A9EF92A5F4A584BF

Seus codepoints em hexadecimal

0B1E0D 13AA27 14A9 F4A5 12513F

E em decimal

728589 1288743 5289 62629 1200447

2. Seja o seguinte conjunto de 5 caracteres UNICODE codificados em UTF-8.

39F6B2B6A503E68FAFF59A84B1

Seus codepoints em hexadecimal

39 1B2DA5 03 63EF 15A131

E em decimal

57 1781157 3 25583 1417521

21.4 Exercício

A seguir um conjunto de 5 caracteres UNICODE codificados em UTF-8. Você deve examiná-los e descobrir quais os 5 codepoints primeiro escritos em hexadecimal e depois em decimal (é só uma simples conversão).

E89A8DC4A9F69C82A1F4ABBABBCA8F

21.4.1 Resposta

1 : 868D 0129 19C0A1 12BE8B 028F

34445 297 1687713 1228475 655

Capítulo 22

Um estudo de otimização

22.1 Otimização de algoritmos em Python

Seja o problema 12 do magnífico site de matemática e programação projecteuler.net. Ele diz: A sequência dos números triangulares é gerada pela adição dos números naturais. Assim, o 7º número triangular será $1+2+3+4+5+6+7 = 28$. Os primeiros 10 números triangulares são

1, 3, 6, 10, 15, 21, 28, 36, 45, 55, ...

Vai-se listar aqui os divisores dos primeiros sete números triangulares:

1: 1

3: 1,3

6: 1,2,3,6

10: 1,2,5,10

15: 1,3,5,15

21: 1,3,7,21

28: 1,2,4,7,14,28

Pode-se ver que 28 é o primeiro número triangular que tem mais do que 5 divisores. Qual é o valor do primeiro número triangular que tem mais do que 500 divisores ?

22.1.1 Primeira estratégia

Sem pensar muito, pode-se resolver a questão usando a força bruta. Aqui, geram-se os números triangulares como descrito e depois acham-se todos seus divisores. A busca acaba quando se acha o primeiro com mais do que 500 divisores. (Uma informação: inteiros de 32 bits são suficientes para este problema). Uma solução em python:

```
t=1
a=1
cnt=0
while cnt <= 500:
    cnt=0
    a=a+1
    t=t+a
    i=1
    while i<=t:
        if t%i == 0:
            cnt=cnt+1
        i=i+1
    print(t)
```

A implementação acima é muito demorada (estimei em umas 5h em uma CPU rápida)

22.1.2 Segunda estratégia

Uma melhora evidente é interromper a busca quando o divisor alcança a raiz quadrada do número triangular: Para cada divisor abaixo da raiz quadrada existe um acima dela. O código fica:

```
t=1
a=1
cnt=0
while cnt <= 500:
    cnt=0
    a=a+1
    t=t+a
    i=1
    ttx = sqrt(t)
    while i<=ttx:
        if t%i == 0:
            cnt=cnt+2
        if t==ttx*ttx:
            cnt=cnt-1 # correção quadrado perfeito
    print(t)
```

Ainda assim, esta segunda estratégia continua muito demorada.

22.1.3 Terceira estratégia

Vamos nos socorrer de um pouco de matemática. Sabe-se que qualquer inteiro N pode ser expresso de maneira única como

$$N = p_1^{a_1} \times p_2^{a_2} \times p_3^{a_3} \times \dots$$

onde p_i é um número primo e a_n seu expoente. Por exemplo, $28 = 2^2 \times 7^1$. Daqui, o número de divisores de N , $D(N)$ de qualquer inteiro pode ser computado como

$$D(N) = (a_1 + 1) \times (a_2 + 1) \times \dots$$

onde os a_n são os expoentes dos fatores primos que compõe o número N . No exemplo, os divisores de 28 são $D(28) = (2 + 1) \times (1 + 1) = 3 \times 2 = 6$. Uma tabela de primos é necessária para aplicar esta relação. Uma estratégia para criá-la pode ser (até 65500):

```
def primo(n):
# devolve True se n primo e False se n composto
    import math
    if n==1:
        return False
    if n<4:
        return True
    if n%2==0:
        return False
    if n<9:
        return True
    if n%3==0:
        return False
    r=math.floor(math.sqrt(n))
    f=5
    while f<=r:
        if n%f==0:
            return False
        if (n%(f+2))==0:
            return False
        f=f+6
    return True
def criatabprimos():
    pri=[2]
    i=1
    j=3
    while j<65500:
        if primo(j):
```

```

        pri.append(j)
        i=i+1
        j=j+2
f=open("f:/p/.../primos.lis","w")
f.write(str(a)[1:-1])
f.close()

```

Depois, para ler essa tabela, usar:

```

def leiapri():
    f=open("f:/p/.../primos.lis","r")
    a=f.read()
    pri=list(map(int,a.split(", ")))
    print("feito")

```

Agora, o processamento

```

t=1
a=1
cnt=0
pri # lista de primos (acima...)
while cnt <= 500:
    cnt=1
    a=a+1
    t=t+a
    tt=t
    i=0
    while i < len(pri):
        if pri[i]*pri[i]>tt:
            cnt=cnt*2
            break
        expoente = 1
        while tt%pri[i]==0:
            expoente=expoente+1
            tt=tt//pri[i]
        if expoente>1:
            cnt=cnt*expoente
        if tt==1:
            break
        i=i+1
print(t)

```

Quarta estratégia

Ainda se pode melhorar mais, sabendo que os números triangulares também podem ser obtidos de acordo com

$$t = \frac{n \times (n + 1)}{2}$$

onde os componentes n e $n + 1$ são necessariamente co-primos (isto é, não têm qualquer fator primo comum nem divisor comum). Aqui, o número de divisores $D(t)$ pode ser obtido

$$D(t) = D(n/2) \times D(n + 1) \text{ se } n \text{ par}$$

ou

$$D(t) = D(n) \times D((n + 1)/2) \text{ se } n + 1 \text{ par}$$

Cada componente é muito menor do que o número triangular. A tabela de primos também é muito menor (contém apenas primos menores do que 1000) Mais, o resultado do componente $n + 1$ pode ser reaproveitado como n no próximo número triangular, sem ser necessário recalculá-lo. Fica:

```

n=3
Dn=2
cnt=0
f=open("f:/p/.../primos.lis","r")
a=f.read()

```

```

prix=list(map(int,a.split(", ")))
pri=prix[0:168] # lista primos até 1000
while cnt <= 500:
    n = n+1
    n1 = n
    if n1 % 2 == 0:
        n1 = n1//2
    Dn1 = 1
    i=0
    while i<len(pri):
        if pri[i]*pri[i] > n1:
            Dn1=2*Dn1
            break
        exponent=1
        while n1 % pri[i] == 0:
            exponent=exponent+1
            n1 = n1//pri[i]
        if exponent > 1:
            Dn1=Dn1*exponent
        if n1 == 1:
            break
        i=i+1
    cnt = Dn*Dn1
    Dn = Dn1
print (int((n*(n-1)/2)))

```

Esta implementação demorou menos de $\frac{1}{10}$ de segundo na mesma CPU lá do começo. Veja-se como vale a pena estudar algoritmos.

22.1.4 Exercício

O objetivo deste exercício é treinar a implementação de algoritmos em python e também meditar sobre como modificações na implementação de um mesmo problema a partir de conhecimentos matemáticos (Santa Matemática !) podem reduzir radicalmente os tempos de processamento.

Para ajudar nesta meditação, implemente a alternativa que quiser e informe qual o número inteiro triangular que tem mais do que

168

divisores.

Resposta

1385280

Capítulo 23

Pacote: Django

É um framework para construir aplicações WEB. Foi criado originalmente como sistema para gerenciar um site jornalístico na cidade de Lawrence, no Kansas. Virou depois um projeto de código aberto.

23.1 Passo a Passo: tutorial 1 da documentação

23.1.1 Obter e instalar os pacotes

Instale o pacote Django

23.1.2 Path

Você precisa fazer o python acessível de qualquer lugar. Assim, sua variável path deve refletir este fato. Para isso:

1. Painel de controle
2. Aba system
3. Aba advanced system settings
4. Procure a variável PATH e acrescente a ela `;%python3` ou o nome onde o python tiver sido instalado
5. Salve e feche tudo

23.1.3 Criar o diretório do projeto Django

Vá em algum lugar (diretório) e crie uma pasta para o projeto. Aqui será

```
cd\  
mkdir lixopro  
cd lixopro  
python c:\python3\scripts\django-admin.py startproject mysite
```

O resultado disso é a criação de um diretório abaixo de lixopro chamado mysite onde o django vai guardar coisas. Eis o que foi criado

```
mysite/  
  manage.py  
  mysite/  
    __init__.py  
    settings.py  
    urls.py  
    wsgi.py
```

cada um desses arquivos é:

- O mysite externo é um container para o seu projeto. O nome dele não interessa.
- `manage.py` é um utilitário de linha de comando que permite a você interagir com o projeto django. Mais em <https://docs.djangoproject.com/en/2.0/ref/django-admin/>
- o mysite interno é o diretório do projeto. Você usará este nome para importar coisas dentro dele (por exemplo `mysite.urls`).

- `mysite/__init__.py` é um arquivo vazio que diz ao python que este diretório deve ser um pacote python.
- `mysite/setting.py` configuração para o projeto django.
- `mysite/urls.py` As declarações de url. É uma tabela de conteúdo.
- `mysite/wsgi.py` Um entry=point para web-servers compatíveis com WSGI.

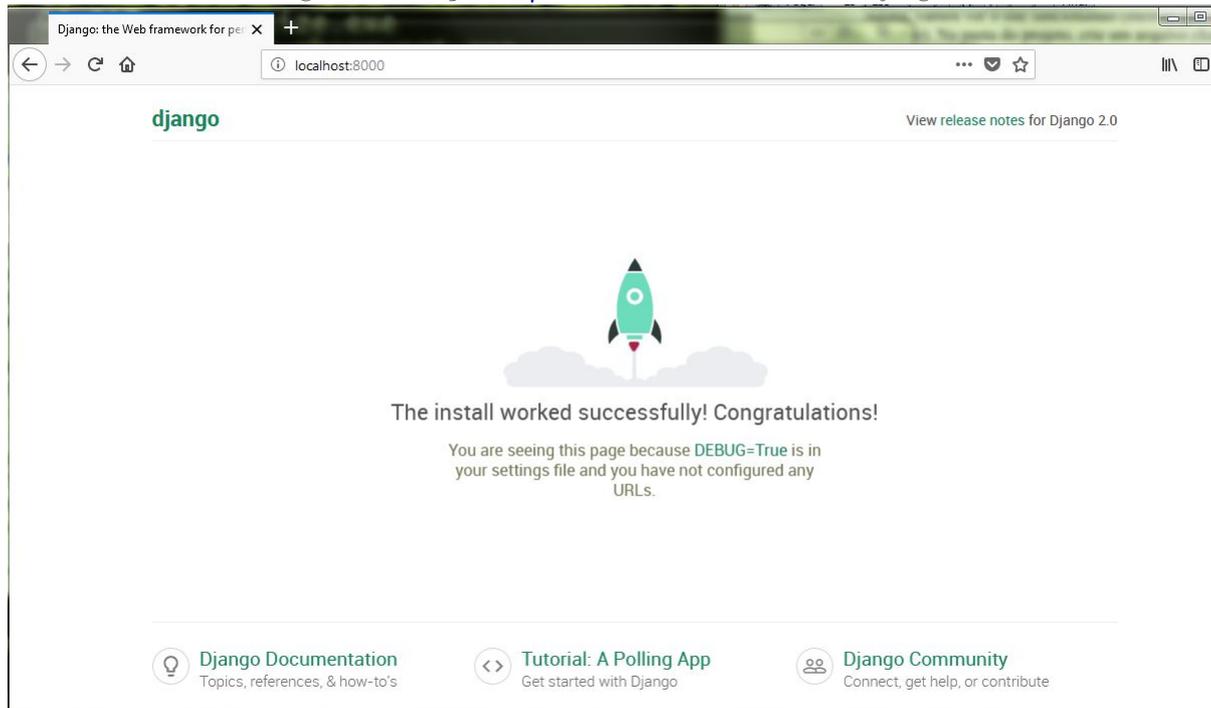
23.1.4 Começando

Vá nesse diretório e execute

```
> python manage.py runserver
```

23.1.5 Testar com o browser

Abra um browser e entregue o endereço <http://localhost:8000/>. O django vai escrever a tela



23.1.6 A qualquer momento

Em qualquer janela dos, faça

```
> python -m django --version
```

e deve obter algo como **2.0.7** indicando a versão do django que está instalada.

23.1.7 Criando sua primeira aplicação

Trata-se de escrever uma APP. A propósito, qual a diferença entre projeto e APP ? Uma APP é uma aplicação web que faz alguma coisa. Um projeto é uma coleção de configurações e APPs para um website particular. Um projeto pode conter múltiplas APPs e uma APP pod ser usada em múltiplos projetos.

Para criar uma app vá para o diretório onde está `manage.py` e digite o comando

```
>python manage.py startapp polls
```

Isto vai criar o diretório polls que deverá ter

```
polls/
  __init__.py
  admin.py
  apps.py
  migrations/
    __init__.py
  models.py
  tests.py
  views.py
```

23.1.8 Escrevendo a primeira coisa

Abra o arquivo `polls/views.py` e coloque algum código python nele

```
from django.http import HttpResponse
def index(request):
    return HttpResponse("Ola mundo cruel")
```

Agora precisamos configurar uma URL Dentro do diretório polls crie um arquivo chamado `urls.py`. Agora seu diretório deve estar assim

```
polls/
  __init__.py
  admin.py
  apps.py
  migrations/
    __init__.py
  models.py
  tests.py
  urls.py      ----> recém criado
  views.py
```

O arquivo `polls/urls.py` deve incluir o seguinte código

```
from django.urls import path
from . import views
urlpatterns=[
    path('', views.index, name='index'),
]
```

O próximo passo é atualizar a url no módulo `mysite/urls.py` Adicione um import para `django.urls.include` e um insert na lista de `urlpatterns`. O arquivo `mysite\urls.py` deve ficar assim

```
from django.contrib import admin
from django.urls import include, path

urlpatterns = [
    path('polls/', include('polls.urls')),
    path('admin/', admin.site.urls),
]
```

23.1.9 Para testar

Entrar no diretório `\lixopro\mysite`

```
> python manage.py runserver
```

no browser `//localhost:8000/polls` e deve aparecer a mensagem

```
alo mundo cruel
```

Isto encerra o equivalente ao tutorial 1 da documentação do Django

23.2 Passo a passo: tutorial 2

23.2.1 Estabelecendo o banco de dados

Deve-se manipular `mysite/settings.py`. É um módulo normal python com variáveis controle do Django. Por default, a configuração usa SQLite. Este já está instalado no Python, assim, você não precisa instalar mais nada.

Se você quiser usar outro banco de dados (mysql ?), instale ele e depois altere os seguintes parâmetros no item DATABASES 'default' de acordo com o banco. Este cara é um dicionário contendo as coisas para interligar com um banco de dados. Eis a definição para SQLite:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': 'mydatabase',
    }
}
```

Eis um exemplo para postgres

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'mydatabase',
        'USER': 'mydatabaseuser',
        'PASSWORD': 'mypassword',
        'HOST': '127.0.0.1',
        'PORT': '5432',
    }
}
```

A primeira coisa para criar um banco de dados é alterar o módulo

```
/pro/mysite/polls/models.py (módulo models.py em polls)
```

```
from django.db import models

class Question(models.Model):
    question_text = models.CharField(max_length=200)
    pub_date = models.DateTimeField('date published')

class Choice(models.Model):
    question = models.ForeignKey(Question, on_delete=models.CASCADE)
    choice_text = models.CharField(max_length=200)
    votes = models.IntegerField(default=0)
```

Deve-se notar que cada entidade deve ser uma classe. Os tipos de campo, (entre outros) podem ser: AutoField, BinaryField, BooleanField, CharField, DateField, DateTimeField, DecimalField, DurationField, EmailField, FileField, FloatField, ImageField, IntegerField, URLField

Os relacionamentos são indicados usando ForeignKey, ManyToManyField, OneToOneField.

Depois de montar seu banco de dados, é necessário dizer ao Django que a aplicação (polls no caso) está instalada. Então em

```
/pro/mysite/mysite/settings.py
```

```
INSTALLED_APPS = [
    'polls.apps.PollsConfig',
    ...
]
```

Agora deve-se rodar uma execução de manage para informar ao Django que foram feitas modificações no banco de dados e ele deve obedecê-las.

```
/pro/mysite
```

```
> python manage.py makemigrations polls
```

Se quiser ver o que este cara fez, execute `python manage.py sqlmigrate polls 0001` Ele vai listar o que irá fazer.

Para fazer a implementação real no banco:

```
/pro/mysite
```

```
> python manage.py migrate
```

Em resumo, para manutenção no banco de dados, são 3 coisas:

1. Mude seus modelos em `/pro/mysite/polls/models.py`
2. Rode em `/pro/mysite` `python manage.py makemigrations`
3. No mesmo local, rode `python manage.py migrate`

Para olhar o ambiente e brincar com ele, deve fazer

```
/pro/mysite
```

```
> python manage.py shell
```

Este cara vai abrir um Python interativo que vai permitir que você brinque com o ambiente. lembrando que a primeira coisa que precisa fazer é importar as classes (entidades) que você definiu no seu modelo. Então:

```
>>> from polls.models import Choice, Question
```

Para sair deste shell, faça `quit()`.

Agora, vai-se lidar com a administração do site. Para criar um superusuário, faça

```
/pro/mysite
```

```
> python manage.py createsuperuser
```

Agora ele vai pedir o nome, email e senha (2 vezes) do sujeito.

Para iniciar o servidor de desenvolvimento faça

```
/pro/mysite
```

```
> python manage.py runserver
```

Depois, você deve abrir um browser e escrever `127.0.0.1:8000/admin/` Deve aparecer uma tela de administração.

Para incluir o novo sistema desenvolvido na tela de administração:

```
/pro/mysite/polls/admin.py
```

```
from django.contrib import admin
from .models import Question
admin.site.register(Question)
```

23.3 Passo a Passo: Tutorial 3

Agora vão ser criadas as interfaces do sistema, as views. Uma view é uma página web que faz algo e tem um template específico.

Para escrever views,

```
em pro/mysite/polls/views.py
```

```
def detail(request, question_id):
    return HttpResponse("You're looking at question %s." % question_id)

def results(request, question_id):
    response = "You're looking at the results of question %s."
    return HttpResponse(response % question_id)

def vote(request, question_id):
    return HttpResponse("You're voting on question %s." % question_id)
```

Agora vá no browser e peça `/polls/1/results/`

Note que este tutorial vai ensinando a criar views e templates, mas não me interessa muito. Vou direto para

23.4 Passo a passo: Tutorial 4

Criando um formulário simples

```
polls/templates/polls/detail.html
```

```
<h1>{{ question.question_text }}</h1>
{% if error_message %}<p><strong>{{ error_message }}</strong></p>{% endif %}
<form action="{% url 'polls:vote' question.id %}" method="post">
{% csrf_token %}
{% for choice in question.choice_set.all %}
<input type="radio" name="choice" id="choice{{ forloop.counter }}" value="{{ choice.id }}" />
<label for="choice{{ forloop.counter }}">{{ choice.choice_text }}</label><br />
```

```
{% endfor %}
<input type="submit" value="Vote" />
</form>
```

Agora uma view que manipula o voto

23.5 Resumão

1. Baixar Python e Django. Anotar os diretórios onde estão. Criar um path que acesse o python.
2. Criar um diretório (md pro) entrar nele e executar `python c:\python\scripts\django-admin startproject mys`
Este cara vai criar uma árvore de diretórios.
3. Para pôr o servidor em ação, vá em `cd/pro/mysite` e execute `python manage.py runserver`. A saída é com `quit()`.
4. Para criar um aplicativo em `/pro/mysite` `python manage.py startapp nome`
5. Altere a view em `/pro/mysite/nome/views.py` incluindo

```
from django.http import HttpResponse
def index(request):
    return HttpResponse("Alo Mundo")
```

6. Em `/pro/mysite/nome` crie `urls.py` e nele inclua

```
from django.urls import path
from . import views
urlpatterns = [
    path('', views.index, name='index'),
]
```

7. Agora, em `mysite/urls.py`, acrescente

```
from django.contrib import admin
from django.urls import include, path
urlpatterns = [
    path('nome/', include('nome.urls')),
    path('admin/', admin.site.urls),
]
```

8. Se tudo funcionou, ao chamar o servidor (`runserver`), depois um browser com `127.0.0.1:8000/nome/` deverá aparecer o alo mundo.
9. Para configurar o banco de dados, atualize `mysite/settings.py`. Se for usar SQLite, nada a fazer.
10. Em `nome/models.py` defina seu banco de dados, a saber

```
from django.db import models
class ENTIDADE(models.Model):
    campo = models.tipo(opções)
    ... = ...
```

Principais tipos: CharField, dateField, DecimalField, FloatField, IntegerField... Relacionamentos: ForeignKey, ManyToManyField e OndeToOneField.

11. Em `mysite/settings.py`, altere `INSTALLED_APPS`, como em

```
INSTALLED_APPS = [
    'nome.apps.NomeConfig',
    'django.contrib.admin',
    ...
]
```

12. Em `/pro/mysite`, execute `python manage.py makemigrations nome`.

13. No mesmo lugar, faça `python manage.py migrate`
14. Para testar e brincar com seus arquivos, em `/pro/mysite` execute `python manage.py shell`. Você abrirá uma sessão python interativa com acesso a seus arquivos.
15. Para incluir registros:

```
x = Entidade(campo = '...', campo=234...)
x.save()
```

16. Em `/nome/models.py`, atualize as classes de cada entidade, incluindo nelas

```
def __str__(self):
    return self.campo descritivo
```

17. Criando um usuário administrador. Vá em `/pro/meusite` e execute `python manage.py createsuperuser`. Depois crie um usuário admin, dê um email e uma senha. A seguir chame o servidor via runserver, e via browser vá para `127.0.0.1:8000/admin/` e entre no usuário admin.

18. Torne a aplicação visível ao admin: Vá em `pro/mysite/nome` e crie um arquivo chamado `admin.py` e nele escreva

```
from django.contrib import admin
from .models import Entidade
admin.site.register(Entidade)
```

Agora, reexecutando o passo anterior, você vai ser sua entidade no admin

19. Em `/pro/mysite/nome/urls.py` faça a modificação

```
from django.urls import path
from . import views
app_name = 'nome'
urlpatterns = [
    path('', views.IndexView.as_view(), name='index'),
    path('<int:pk>/', views.DetailView.as_view(), name='detail'),
    ...
]
```

20. Em `/pro/mysite/nome/views.py`, altere

```
from django.http import HttpResponseRedirect
from django.shortcuts import get_object_or_404, render
from django.urls import reverse
from django.views import generic
from .models import Entidade, Entidade
class IndexView(generic.ListView):
    template_name = 'nome/index.html'
    context_object_name = ...
    def get_queryset(self):
        return ...
class DetailView(generic.DetailView):
    model=Entidade
    template_name = 'nome/detail.html'
...

```

21. Crie formulários. Em `pro/mysite/nome/templates/nome` crie um `detail.html` contendo

```
<h1>{{ question.question_text }}</h1>
{% if error_message %}<p><strong>{{ error_message }}</strong></p>{% endif %}
<form action="{% url 'polls:vote' question.id %}" method="post">
{% csrf_token %}
{% for choice in question.choice_set.all %}
<input type="radio" name="choice" id="choice{{ forloop.counter }}" value="{{ choice.id }}" />
<label for="choice{{ forloop.counter }}">{{ choice.choice_text }}</label><br />
{% endfor %}
<input type="submit" value="Vote" />
</form>
```

-
22. Crie arquivos estáticos. Em `/pro/mysite/nome/static/nome/` crie o arquivo `style.css` e nele inclua o estilo desejado para o site.

E daí, em `/pro/mysite/nome/template/nome/index.html` faça a referência ao estilo criado escrevendo

```
{% load static %}
<link rel="stylesheet" type="text/css" href="{% static 'nome/style.css' %}" />
{% if latest_question_list %}
  <ul>
    ...
```

23. Se quiser criar imagens... Em `/pro/mysite/nome/static/nome/images/` crie uma imagem chamada `background.gif`

E depois, no estilo acima, faça

```
li a {
  color: green;
}
body {
  background: white url("images/background.gif") no-repeat;
}
```

Capítulo 24

Tradução Automática

tradução O python pode chamar o Google Translator e traduzir textos de maneira automática. Não há necessidade de autenticação ou registro. É muito simples. O módulo é

```
pip install translate
```

Parece ter havido uma evolução para o pacote `googletrans`. Deve ser importado igual (`pip install googletrans`) e depois chamado igual (`from googletrans import Translator`)

Daí para a frente segue igual...

Instalado o módulo, basta chamá-lo

```
>>> from translate import Translator #ou from googletrans import Translator
>>> translator=Translator(to_lang='pt')
>>> translation=translator.translate('Hello world, how are you?')
>>> print(translation)
Olá mundo, tudo bem?
>>> translator=Translator(to_lang='es')
>>> translation=translator.translate('Hello world, how are you?')
>>> print(translation)
Hola mundo como estas
>>> translator=Translator(to_lang='la') #latim
>>> translation=translator.translate('Hello world, how are you?')
>>> print(translation)
Orbis terrarum salve, quid agis?
>>> translator=Translator(to_lang='fr')
>>> translation=translator.translate('Hello world, how are you?')
>>> print(translation)
bonjour monde comment vas tu
>>> translator=Translator(to_lang='ca') # catalão
>>> translation=translator.translate('Hello world, how are you?')
>>> print(translation)
Hola món, com estàs?
>>> translator=Translator(to_lang='eu') # aparentemente é BASCO
>>> translation=translator.translate('Hello world, how are you?')
>>> print(translation)
Kaixo mundua, nola zaude?
```

Um programa meio completo de tradução de um arquivo

```
import codecs
from translate import Translator
def lear():
    translator = Translator(from_lang='pt', to_lang='fr')
    arq=codecs.open('tlixo.txt','rb', encoding='utf-8')
    for linha in arq:
        linha=linha.rstrip()
        translation = translator.translate(linha)
        print(translation)
    arq.close()

lear()
```

Os tradutores tem limites para contas anônimas (free). O mymemory.translated.net (default) tem um limite de 1000 palavras por dia. A microsoft também tem o serviço (<http://docs.microsofttranslator.com/>), mas também tem limites restritos para contas anônimas.

Os códigos de idioma são:

auto	0 google descobre a origem
pt	português
es	espanhol
ar	arábico
ca	catalão
cs	checo
de	alemão
el	grego
en	inglês
eo	esperanto
eu	basco
gn	guarani
it	italiano
ja	japonês
ko	coreano
nl	holandês
tl	tagalog
zu	zulu

veja a lista completa em [https://kodi.wiki/view/List_of_language_codes_\(ISO-639:1988\)](https://kodi.wiki/view/List_of_language_codes_(ISO-639:1988))

Capítulo 25

Leitura e geração de arquivos PDF

Como se sabe o padrão PDF nasceu como uma iniciativa da Adobe para uniformizar o acesso a dispositivos de impressão ¹. A Adobe lançou dois produtos pagos: o gerador e o leitor de PDF. Lentamente o formato começou a se popularizar e em dado momento o Serviço de Rendas americano comprou uma licença genérica do leitor para que todos os contribuintes americanos pudessem ler arquivos PDF. Só a economia de papel, pagou com sobras tal licença. A partir desse ponto, foi como se a licença tivesse se tornado pública. O fabricante chegou a essa conclusão e liberou tudo. Mais tarde, essa liberação foi oficializada e hoje o padrão PDF é público e livre. Há uma estatística interessante aqui: Se os manuais do Boeing 747 que hoje são distribuídos em PDF fossem impressos em papel, seriam necessários 5 aviões para carregar a documentação de um único avião.

Padrão PDF

O padrão *Portable Document Format (PDF)* é o padrão de fato em descrição de páginas impressas, e corre a passos largos para se tornar a maneira pela qual vastos documentos são e serão produzidos de agora para o futuro. É comum hoje, ao comprar algum artefato tecnológico que exija documentação para sua operação, que o mesmo venha acompanhado de um CD ao invés de uma caixa de manuais. É óbvio que o CD conterá os mesmos (ou mais) manuais escritos em formato PDF.

Um arquivo PDF descreve como uma página ficará depois de impressa e ele é melhor do que imagem dessa página pelas seguintes razões

- A imagem como tal sempre é muito maior do que sua descrição. A razão para isso é a evidente redundância de qualquer imagem.
- A imagem como tal é fixa e só pode ser rotacionada, redimensionada ou de alguma maneira modificada à custa de processamento e perda de qualidade.
- Sobre uma imagem não há como manipular o conteúdo, por exemplo conduzindo uma busca textual (⌘F alguma coisa ou então ⌘C e ⌘V).
- Alterar uma única palavra sobre uma imagem exige habilidades de pintura. Sobre a descrição de uma página é tarefa simples: basta trocar uma palavra pela outra.
- Imagine uma imagem enviada para um monitor (72 DPP) ou a mesma imagem enviada para uma impressora (300 DPP). Não podem ser a mesma imagem.

A versão 1.0 do padrão PDF nasceu em 1993 quando a Adobe lançou 2 programas: o Distiller (para gerar PDF) e o Reader (para ler): ambos pagos. A coisa começou a mudar quando o departamento de rendas americano comprou uma licença que permitia aos contribuintes baixar o Reader. Rapidamente a Adobe deu-se conta de que seria melhor para ela distribuir gratuitamente o Reader. Nos dez anos seguintes o PDF acabou se tornando o padrão de fato de descrição de imagens.

Apresenta as seguintes vantagens:

- acesso aleatório: qualquer página pode ser montada independente das demais.
- linearização: todos os elementos necessários para montar uma página estão juntos.
- atualização incremental: mudanças ficam registradas no fim do arquivo. Vantagem: opção de desfazer ilimitada e rápida atualização. Desvantagem: o arquivo não pára de crescer.

¹Só para o leitor ter idéia de como este era um problema e tanto no final dos anos 80, lembro que ao receber um pacote com o software Word, ainda na sua versão 2, na caixa havia mais de 15 disquetes. Dois deles continham o software. Todos os demais continham drivers de instalação para centenas de impressoras que havia pelo mundo. E havia algumas impressoras brasileiras que não apareciam. Era a época da reserva de mercado.

- fontes incorporadas: o destino sempre será montado corretamente. A fonte é desbastada de tudo que não é necessário.
- padronização ISO: em 2008 a ISO abraçou o padrão Adobe PDF 1.7, o que o transformou em padrão aberto.
- compatibilidade para trás e para a frente: Qualquer leitor lê as versões antigas e deve ler as futuras, já que todos os leitores ignoram o que não conhecem.
- integrado a todos os principais browsers do protocolo [HTTP](http://).

Um arquivo PDF

Cabeçalho Formado por 2 linhas: a primeira identifica o arquivo como um PDF e informa a versão PDF em que ele foi gerado. A segunda linha inclui caracteres que não podem ser impressos (para avisar eventuais leitores deste fato). Exemplo:

```
%PDF-1.0
%âãõα
```

Corpo Um conjunto de nodos de um grafo, contendo as páginas, conteúdo gráfico, textos, sempre na forma de objetos. Cada objeto começa com um número de objeto (iniciando em 1), um número de geração (sempre 0) e a palavra chave **obj**. O objeto termina pela palavra chave **endobj**. Exemplo:

```
1 0 obj      -- descrevendo o objeto 1
<<          -- começa um dicionário
  /Kids [2 0 R]  -- /Kids tem o valor 2 0 R
  /Count 1      -- só um
  /Type /Pages  -- do tipo : página
>>          -- fim do dicionário
endobj       -- fim do objeto
```

Tabela de Referência Cruzada Lista o deslocamento em bytes de cada objeto no corpo do arquivo. Exemplo

```
0 6          -- seis entradas na tabela
0000000000 65535 f  -- entrada especial
0000000015 00000 n  -- obj1 no desloc 15
0000000074 00000 n  -- obj2 no desloc 74
...
```

Rodapé Informações e metadados para aumentar a performance de acesso. A primeira entrada é **trailer**. Depois o dicionário de trailer que deve apresentar ao menos a entrada **\Size** dizendo quantos itens há na tabela de referência cruzada e **\Root** dizendo qual objeto é o catálogo do documento. Depois vem uma linha que só contém **startxref** seguida por uma única linha indicando o deslocamento em bytes utilizado no início da tabela de referência cruzada do arquivo. Termina tudo a linha **%%EOF**. Exemplo:

```
trailer      -- palavra chave
<<          -- começa um dicionário
  /Root 5 0 R  -- o catálogo é o objeto 5
  /Size 6      -- tem o tamanho 6
>>          -- fim do dicionário
startxref
459         -- deslocamento da tabela de ref cruzada
%%EOF      -- fim de arquivo
```

Componentes

Uma descrição PDF suporta 5 componentes básicos: i. Números inteiros e reais (mas não exponenciais); ii. strings que são escritos entre parênteses; iii. Nomes, que sempre começam por uma barra normal; iv. Os valores booleanos **true** e **false** e v. O componente nulo (**null**). Aceita também 3 componentes compostos: i. Arrays, que são uma coleção ordenada de outros componentes, escritos entre colchetes; ii. Dicionários que são uma lista não ordenada de duplas: nome e componente. Começam por **<<** e terminam por **>>**. Finalmente iii. Fluxos, que armazenam dados binários, sempre acompanhados de um dicionário que descreve seus atributos, como comprimento e parâmetros de compactação, por exemplo.

Exemplo de um arquivo PDF

Eis um exemplo simples de um arquivo PDF não comprimido.

```
%PDF-1.0
1 0 obj
<< /Type /Pages
/Count 1
/Kids [2 0 R]
>>
endobj

2 0 obj
<< /Type /Page
/Mediabox [0 0 612 792]
/Resources 3 0 R
/Parent 1 0 R
/Contents [4 0 R]
>>
endobj

3 0 obj
<< /Font
<< /F0
<< /Type /Font
/BaseFont /Times-Italic
/Subtype /Type1 >>
>>
>>
endobj

4 0 obj
<< >>
stream
1. 0. 0. 1. 50. 700. cm
BT
/F0 72. Tf
(Ola Mundo!) Tj
ET
endstream
endobj

5 0 obj
<< /Type /Catalog
/Pages 1 0 R
>>
endobj

6 0 obj
<<
/Author (...)
/Keywords (teste de um pdf feito a mao)
>>
endobj

xref
0 7
trailer
<<
/Size 7
/Root 5 0 R
/Info 6 0 R
>>
startxref
```

0
%%EOF

Utilitários de manipulação O pdftk possui as seguintes funcionalidades: mesclar vários pdf em um só; comprimir e descomprimir arquivos; quebrar um arquivo em vários; incluir marcas d'água, criptografar e descriptografar entre outras finalidades.

Referência para o formato PDF:

www.adobe.com/devnet/acrobat/pdfs/pdf_reference_1-7.pdf

25.1 Capturando texto em PDF

Como tudo na vida, a generalidade tem um preço em termos de funcionalidade. Dizendo melhor: dispositivos genéricos são mais difíceis de usar do que dispositivos específicos.²

Como o PDF aceita igualmente bem entradas formadas por imagens e por textos, indistinguindo-os na saída, deve-se fazer a ressalva que a habilidade a ser estudada a seguir só é válida se o arquivo PDF que vai ser lido tiver usado textos para montar a saída. Se ele usou imagens, tudo o que vem a seguir é inútil e não vai funcionar. Isto é muito comum quando se escaneia um texto e o resultado é publicado como PDF. Lá só há imagens.

Para ler e decodificar um arquivo PDF, deve-se instalar o módulo **pyPDF2** através de

```
\python\scripts> pip install pyPDF2
```

Dentro do Python, deve-se emitir

```
>>> from pyPDF2 import PdfFileReader
```

Abrir o arquivo exemplo.PDF e criar um objeto PDF dentro do Python

```
>>> fil = open('exemplo.PDF', 'rb')
>>> docu = PdfFileReader(fil)
```

Obter o número de páginas e verificar se ele está ou não criptografado

```
>>> docu.numPages
5
>>> docu.isEncrypted
False
```

Se ele estivesse criptografado, antes de recuperar qualquer informação será necessário oferecer a senha

```
>>> docu.decrypt('senha')
1
```

Note que o PyPDF2 só consegue lidar com alguns métodos de criptografia (na verdade o 1 e o 2). Se outro método foi usado a descriptografia não vai funcionar e então será necessário pré-processar o PDF descriptografando-o antes de submetê-lo a este programa Python.

Obter a primeira página e recuperar o texto nela. O texto vem como uma lista de strings

```
>>> docu.pages[0].extractText()
'Este documento foi escrito e produzido...'
```

Ao final, o arquivo deve ser fechado

```
>>> fil.close()
```

²Se o leitor discorda, pense em 2 controles remotos: um deles serve para qualquer aparelho e o outro é específico de um toca discos. Qual será mais simples de usar ?

Capítulo 26

Gerando arquivos PDF

Há uma sutil mudança aqui: enquanto sistemas antigos geravam extensas saídas diretamente em papel, hoje – até por questões ecológicas – essas mesmas saídas tendem a migrar para outras plataformas, sendo o PDF a preferida delas. Assim, vai-se ver aqui como gerar arquivos PDF diretamente do Python.

O pacote a carregar é `fpdf`, assim

```
>>> import fpdf
```

A seguir deve-se criar um documento PDF

```
>>> docu=fpdf.FPDF()
```

Depois estabelecer a fonte e a cor do que vai se escrever

```
>>> docu.set_font('Times','B',18)
>>> docu.set_text_color(20,100,200)
```

Agora, cria-se uma página

```
>>> docu.add_page()
```

A sequência agora é preencher a página. Os comandos são `cell` e `multi_cell`. A diferença é que o `multi` muda de linha quando o final da folha é atingido, enquanto o `cell` não faz isso. Em ambos, os primeiros parâmetros são largura e altura da linha. Uma largura de 0 manda usar a largura da folha. Um valor de 5 (mm) é suficiente para fontes de tamanho 12. A chamada a `ln()` muda de linha. Note que no ambiente `multi`, se for necessária uma nova página o método `multi` faz isso de maneira automática sem que o usuário tenha que se preocupar.

```
>>> docu.cell(0,5,'EXEMPLO2 DE ARQUIVO PDF')
>>> docu.ln()
>>> docu.multi_cell(0,5,'Vamos ver agora' * 20)
[]
```

Finalmente, há que se gravar o arquivo

```
>>> docu.output('arquivopdf2.pdf')
```

Para organizar o relatório (incluir cabeçalhos e rodapés fixos, criar links para partes do documento, sumário, etc) veja a referência [Bul20], págs 183 e seguintes.

Capítulo 27

LATEX: Inclusão de arquivos PDF

Para quem está lidando com LATEX, vão algumas dicas de como arquivos PDF podem ser incluídos dentro de arquivos TEX para formar um pacote completo. Este material mesmo, inclui no final diversas cartelas que eu produzi e que podem ser separadas deste material. Para tanto, é necessário incluir o módulo

```
\usepackage{pdfpages}
```

e depois dentro do texto, emitir o comando

```
\includepdf[pages={in-fi}]{arquivo.pdf} ou  
\includepdf[pages={a,b,c,d}]{arquivo.pdf}
```

Naturalmente, há dezenas de opções distintas para este pacote, que devem ser pesquisadas no CTAN.

A seguir um exemplo prático. Seguem-se cartelas de Symbolic Python (2 páginas), Numerical Python (2 páginas), Matplotlib (idem) e a própria linguagem Python (2 páginas). Os arquivos são incluídos com

```
\includepdf[pages={1-2}]{/p/python/cart2_sympy.pdf}  
\includepdf[pages={1-2}]{/p/python/cart2_numpy.pdf}  
\includepdf[pages={1-2}]{/p/python/cart2_matplotlib.pdf}  
\includepdf[pages={1-2}]{/p/python/cart2_python.pdf}
```


Capítulo 28

Análise de texto

O grande sonho da Inteligência Artificial: não nos esqueçamos que o primeiro desafio dela ainda na década de 1950 foi construir um tradutor automatizado. A tarefa revelou-se muito difícil, mas pode-se afirmar que no século 21, os passos dados nesta direção foram enormes. Aí está o Google Translator que não nos deixa mentir. O Python está devidamente apetrechado para trabalhar fundo nesta questão, mas vale a ressalva: apenas em idioma inglês, ou muito preferencialmente em inglês. Duas explicações: o inglês é hoje o idioma melhor trabalhado na ciência (e em particular na ciência da computação) e também o inglês apresenta certas facilidades inexistentes em linguas latinas (como o português e o espanhol: em particular as concordâncias nominais, verbais, de gênero e grau, que são muito mais complexas no latim).

Um bom local para obter textos em inglês puro (sem formatação) é o projeto gutenberg (<http://www.gutenberg.org>) e em particular o livro 1513 que vem a ser Romeu e Julieta de Shakespeare. O site não permite o processamento de seus conteúdos por programas, apenas por humanos. Olhem o Turing dando as cartas. Razão pela qual deve-se acessar o site e guardar localmente o arquivo. Em particular deve-se retirar dele o início e o final, já que (ao final) aparece a extensa licença de uso, e certamente Shakespeare não escreveu isso.

Para analisar o texto, deve-se fazer

```
\python\3x\scripts> pip install textblob
\python\3x\scripts> python -m textblob.download_corpora
```

O segundo comando carrega diversos componentes a saber:

- o corpus da Universidade Brown para identificar os componentes do discurso
- o tokenizador **punkt**
- o conjunto de definições, sinônimos e antônimos **WordNet**
- o **Averaged Perceptron Tagger**
- o **Conll2000** para quebrar o texto em seus componentes. O nome conll200 vem do criador original dos dados: Conference on Computational Natural Language Learning, 2000.
- Comentários sobre filmes para análise de sentimento

A extensa documentação original do pacote pode ser acessada em <https://textblob.readthedocs.io/en/dev/>
Veja algo aqui

```
>>> a="I don't like very much APL, but Python is better"
>>> b=TextBlob(a)
>>> b
TextBlob("I don't like very much APL, but Python is better")
>>> b.sentiment
Sentiment(polarity=0.38, subjectivity=0.38)
>>> b.words
WordList(['I', 'do', "n't", 'like', 'very', 'much', 'APL', 'but', 'Python', 'is', 'better'])
>>> a='Eu gosto muito de APL'
>>> b=TextBlob(a)
>>> b
TextBlob("Eu gosto muito de APL")
>>> b.detect_language()
'pt'
>>> b.sentiment
Sentiment(polarity=0.0, subjectivity=0.0) # não há análise em português
```

```
>>> sp=b.translate(to='es')
>>> sp
TextBlob("Me gusta mucho APL")
>>> sp.detect_language()
'es'
>>> fr=b.translate(to='fr')
>>> fr
TextBlob("J'aime vraiment APL")
>>> from textblob import Word
>>> ma=Word('apple')
>>> ma.pluralize()
'apples'
>>> ca=Word('campi')
>>> ca.singularize()
'campus'
>>> ani=TextBlob('dog cat fish bird').words
>>> ani.pluralize()
WordList(['dogs', 'cats', 'fish', 'birds'])
>>>
```

o sentimento é um valor variando entre -1 e +1, indicando negatividade ou positividade do sentimento. Já a subjetividade varia entre 0 (completa assertividade) e 1 (totalmente subjetivo).


```

wc2 = WordCloud(stopwords=sw, mask=wcm, background_color='white').generate(texto)
plt.imshow(wc2)
plt.show()

```

```
faz('/p/python/pyt01.tex')
```

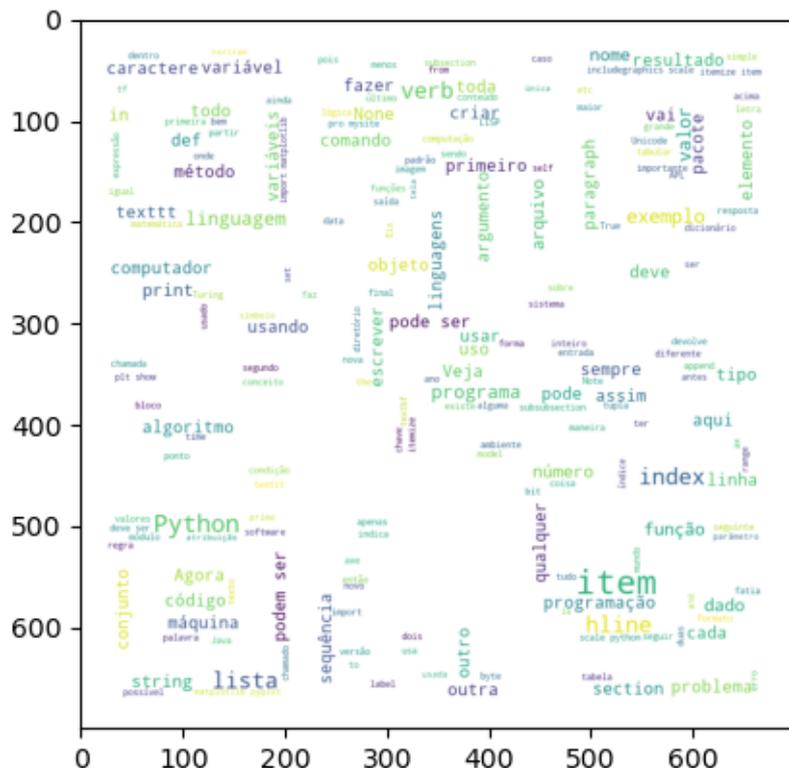
Valem as seguintes observações

- na imagem base, pixels pretos são usados para escrita, enquanto pixels brancos são mantidos vazios. Veja aqui a imagem base



- a codificação UTF8 é necessária para textos escritos em português. Sem ela, há erro na leitura.
- o método `stopwords.words('portuguese')` retorna uma lista (bastante satisfatória) de palavras irrelevantes em português. A lista pode ser estendida, como se vê acima.

com o seguinte resultado



Capítulo 30

Serviços Watson

Watson é um ambiente de aprendizado que a IBM desenvolveu ao longo dos últimos anos. Ele ficou famoso ao ganhar uma disputa do programa Jeopardy (algo parecido ao passa ou repassa do Silvio Santos, mas um pouco melhor no conteúdo) valendo 1 milhão de dólares que a IBM doou para a caridade. Watson respondeu corretamente perguntas consultadas em 200 milhões de páginas, incluindo a wikipedia. É uma plataforma de computação cognitiva baseada na nuvem que simula a capacidade humana de auto aprendizado, à medida em que é utilizada.

30.1 Abrindo uma conta

login perrette silva email perrettesil98@gmail.com senha: perrette98sil@5

Capítulo 31

Pillow: manuseando imagens

Este pacote tem uma história antiga. Ele descende do pacote PIL (python image library) e sendo uma bifurcação (fork) do pacote original, ele herdou a base, de modo que pode haver alguma confusão sobre quem está sendo usado. Uma boa resposta é: os dois.

A principal vantagem aqui é a tratativa de inúmeros tipos de imagens cada um com as suas especificidades e que do ponto de vista do python são uniformizados. A leitura e gravação de todos os tipos é muito fácil, e a conversão de um tipo a outro mais fácil ainda. Fala-se aqui de BMP, PNG, GIF, JPG, TIF entre outros.

Um detalhe é que a apresentação de imagens não é responsabilidade do pacote. Seu método `xxx.show()`, grava a imagem em disco e chama alguma ferramenta do sistema operacional para apresentá-lo.

31.1 Instalação

Como sempre, chame

```
pip install pillow
```

A seguir, alguns exemplos

```
from PIL import Image #1
im=Image.open("c:/p/python/image1.jpg") #2
im.show() #3
im=im.rotate(45) #4
im.save("c:/p/python/image145.bmp") #5
```

#1 importa-se o pacote Image (note a maiúscula) a partir do PIL (e não do pillow)

#2 Abre-se a imagem cujo endereço aparece. Note que as barras são à direita, e que o tipo da imagem é recuperado diretamente de seu nome (.jpg aqui)

#3 A imagem é mostrada (por vias transversas) para garantir que é a que queremos

#4 Ela é girada 45° anti-horário

#5 Transformada em BMP (apenas se sabe disso pelo nome (.bmp aqui) e guardada em disco com outro nome.

Este exemplo simples abre uma avenida imensa: a criação de processadores em batch para grandes quantidades de imagens sem a necessária participação do operador: isso vale ouro.

31.2 Atributos da imagem

```
from PIL import Image
im=Image.open("c:/p/python/ima/mamaei.tif")
print(im.filename) # deu: c:/p/python/ima/mamaei.tif
print(im.format) # deu TIFF
print(im.mode) #deu P
print(im.size) # deu (823, 569)
print(im.width) #deu 823
print(im.height) # deu 569
print(im.info) # deu {'compression': 'raw', 'dpi': (96, 96)}
print(im.palette) #deu <PIL.ImagePalette.ImagePalette
object at 0x00000232A2CC07C0>
```

31.3 Separação e junção

O método `xx.split()` pega a imagem `xx` e separa os canais que a formam. Veja

```
from PIL import Image
xx1=Image.open("c:/p/python/ima/campanar.jpg")
print(len(xx1.split())) # deu 3 - RGB
xx2=Image.open("c:/p/python/ima/campanarc.jpg")
print(len(xx2.split())) # deu 1
xx3=Image.open("c:/p/python/ima/campanari.tif")
print(len(xx3.split())) # deu 1
xx4=Image.open("c:/p/python/ima/gato.png")
print(len(xx4.split())) # deu 3
xx5=Image.open("c:/p/python/ima/lixo1.gif")
print(len(xx5.split())) # deu 1
```

Depois, para juntar o método é `yy=Image.merge("RGB", (r,g,b))` Neste exemplo o modo é RGB, há muitos outros como

1 (1-bit pixels, black and white, stored with one pixel per byte)

L (8-bit pixels, black and white)

P (8-bit pixels, mapped to any other mode using a color palette)

RGB (3x8-bit pixels, true color)

RGBA (4x8-bit pixels, true color with transparency mask)

CMYK (4x8-bit pixels, color separation)

YCbCr (3x8-bit pixels, color video format)

LAB (3x8-bit pixels, the L*a*b color space)

HSV (3x8-bit pixels, Hue, Saturation, Value color space)

I (32-bit signed integer pixels)

F (32-bit floating point pixels)

tem muitos mais, veja na documentação <https://pillow.readthedocs.io/en/stable/reference/...>

Um truque legal aqui é fazer

```
from PIL import Image
im=Image.open("c:/p/python/ima/piscina.jpg") # imagem truecolor RGB
r,g,b = im.split()
xx=Image.merge("RGB", (b,b,r))
xx.show()
```

As cores ficam bagunçadas, já que no merge, ao invés de r,g,b chamou-se com b,b,r.

31.4 Aplicando filtros

Existem diversos filtros dentro do pacote todos chamados de maneira parecida. Os filtros são

BLUR Borra a imagem usando a média de valor dos vizinhos para cada pixel.

CONTOUR Traça as linhas de contorno do desenho: fundo branco e contorno preto

DETAIL Melhora a nitidez da imagem

EDGE_ENHANCE Mais nitidez

EDGE_ENHANCE_MORE Mais mais nitidez

EMBOSS Cria uma imagem como se fosse um alto relevo do original, meio monocromático

FIND_EDGES Imagem em preto com os contornos em branco

SHARPEN Melhora a nitidez

SMOOTH Borra a imagem

SMOOTH_MORE Borra mais a imagem

Para todos eles, a chamada é a mesma

```
from PIL import Image, ImageFilter
from PIL.ImageFilter import (BLUR, CONTOUR, DETAIL, EDGE_ENHANCE_MORE,
                             EMBOSS, FIND_EDGES, SHARPEN, SMOOTH)
im=Image.open("c:/p/python/ima/cidade.jpg")
im.show()
i1=im.filter(SMOOTH) # aqui se troca o nome do filtro
i1.show()
```

31.4.1 Cores

O pacote PILLOW trata cores de diversos formatos arranjados em tabelas o que permite diversos tipos de conversão. A maneira de representação de cores pode ser

hexadecimal no formato **#rrggbb**. Por exemplo vermelho é **#FF0000** e amarelo é **#FFFF00**.

cilíndrico também conhecido como HSL: hue (cor), saturação (força) e iluminação (branco/preto)

nomes de cores Há cerca de 140 nomes de cores padrão suportados pelos browsers. Os nomes são case insensitive.

A seguir a criação de uma imagem nova de 256,256 pixels, com a cor base **#add8e6** (azul claro)

```
from PIL import Image, ImageColor
img=Image.new("RGB", (256,256), ImageColor.getrgb("#add8e6"))
img.show()
```

31.4.2 módulo ImageDraw

Serve para criar imagens a partir de comandos básicos, retocar imagens ou anotar algum texto. A origem de uma imagem (o pixel 0,0) é o canto superior esquerdo da imagem. A seguir a colocação de um "X" vermelho e verde no meio de uma imagem

```
import sys
from PIL import Image, ImageDraw
im=Image.open("c:/p/python/ima/por_sol.jpg")
dra=ImageDraw.Draw(im)
dra.line((0,0)+im.size, fill='#ff0000')
dra.line((0,im.size[1], im.size[0],0), fill='#00ff00')
im.show()
```

Há ainda comandos para elipse, retângulo, polígono, além de escrita de textos e construção de sequências de imagens (veja a documentação).

31.4.3 Imagem com numpy

Aqui o pulo do gato entre imagens e matrizes numéricas. Veja-se o exemplo

```
from PIL import Image
import numpy as np
ar=np.zeros([150,250,3], dtype=np.uint8)
ar[:, :100]=[255,255,0]
ar[:, 100:]=[0,0,255]
im=Image.fromarray(ar)
im.show()
```

cria uma imagem com um pedaço amarelo e outro azul. O caminho de volta:

```
from PIL import Image
from numpy import array
im=Image.open("c:/p/python/ima/moca_agua.jpg")
im2ar=array(im)
im2ar[0, :, 0]=255
ar2im=Image.fromarray(im2ar)
ar2im.show()
```


Capítulo 32

Gerando aplicativo standalone

Como se sabe o Python é interpretado. Alguns dirão que é defeito, outros que é qualidade, mas como tudo na vida, há 2 lados nessa moeda. De um lado, não é fácil gerar um módulo executável pequeno que possa ser enviado para o outro lado do mundo e lá faça o seu papel lindamente. Lembro que esta é uma das mais maravilhosas características do C (C++): a possibilidade de gerar um utilitário *chiquitito pero cumplidor*. Por outro lado, alguma coisa interpretada sempre é melhor de usar, apreender e depurar. Mais: por ser filho do século XXI, Python perdeu de vez a vergonha de usar e abusar da CPU. Para que economizar, se temos recursos de sobra.

Seja como for, a distribuição de produtos Python apresenta um desafio

- A exigência do usuário receptor (por hipótese do outro lado do mundo) tenha um ambiente Python compatível com aquele onde o aplicativo foi gerado. Isto é dificultado pelos inúmeros pacotes e dependências que o Python prolifera. Note que este estado de coisas é solenemente desprezado – por exemplo – pelo Tensorflow, que é a ferramenta de aprendizagem do Google (redes neurais). Todo usuário usa sem se preocupar com este detalhe (a necessidade de ambiente Python) simplesmente porque quem vai usar o TensorFlow deve saber Python de cor. Neste caso vai apenas o programa fonte, o receptor que se vire.
- A possível necessidade de instalar o ambiente Python na máquina destino. Isto é ruim, se o destinatário achar que Python é uma cobra gorda.
- Empacotar e mandar (de maneira transparente) tudo o que é necessário. O problema aqui é o tamanho do pacote que vai.

Note que ao mandar o programa fonte, cria-se nova discussão: o receptor pode alterar o programa recebido e se isso é bom (e geralmente é), por outro lado complica sobremaneira questões como suporte e responsabilidade contratual do fornecedor.

Uma possível solução para esta dificuldade, é o uso do pacote **pyinstaller** que vai pela terceira hipótese aí de cima: vai ser gerado um único executável imenso (já que ele está levando o Python) mas é um único executável que não permite alteração.

Uma questão adicional é que se um mesmo fonte Python roda em qualquer ambiente (Linux, windows ou apple), isto deixa de ser verdade quando se manda um executável: faz todo o sentido pela dependência que agora há em relação ao sistema operacional destino. Se o mesmo programa tiver que rodar em várias plataformas, há que gerar o executável específico (a partir do mesmo fonte) para cada sistema operacional. Este exemplo é para Windows.

32.1 Receita de bolo em windows

1. Instale o pacote **pyinstaller**. Para tanto vá no diretório **/python/scripts** e execute

```
pip install pyinstaller
```

2. Desenvolva o seu programa como sempre. Guarde-o junto com seus módulos em um diretório específico, digamos **/meupro**. Teste-o no seu ambiente (... F5...) até ele estar pronto para ser distribuído.
3. Numa janela CMD, entre nesse diretório e execute

```
\python38\scripts\pyinstaller meupro.py
```

Depois de muita verborragia, o pyinstaller vai criar, abaixo de onde você está, dois subdiretórios: **dist** que contém o executável que é o nosso objetivo e o **build** que pode ser descartado, mas que tem algumas coisas interessantes, a saber: **xref-modulo.html** contém uma lista em HTML de quem importa quem. **warnmodulo.txt** avisos e erros na geração do executável, entre outras.

O diretório `dist` inteiro precisa ser despachado. Junto vão muitas bibliotecas do Python necessárias lá no destino para rodar seu aplicativo.

Uma solução mais radical é gerar UM ÚNICO executável que contenha tudo: basta usar a chave `-F` na geração. O comando agora é `pyinstaller -F meupro.py`. Mas, segure-se: o programa Python abaixo

```
import numpy as np
a=input("Informe o peso da pessoa em kilogramas ")
a=float(a)
b=input("Informe a altura em metros ")
b=float(b)
i=a/b**2
print("Seu IMC é de ",i)
print("tecle algo para encerrar")
```

gerou um executável de 37 megabytes...

Detalhe importante: o mesmo programa mas sem o import do pacote numpy gerou um executável de 7.5 Megabytes.

Já o programa abaixo, que imprime uma watermark em uma figura qualquer

```
from PIL import Image, ImageDraw, ImageFont
no=input("Informe o nome do arquivo ")
te=input("Informe o texto da watermark ")
im=Image.open(no)
wi,he=im.size
draw=ImageDraw.Draw(im)
font=ImageFont.truetype('arial.ttf', 78)
tew,teh=draw.textsize(te,font)
mar=125
x=wi-tew-mar
y=he-teh-mar
draw.text((x,y),te,font=font)
#im.show()
aa=list(no.partition('.'))
aa[0]=aa[0]+'w'
bb=''.join(aa)
im.save(bb)
print("Salvo com nome original mais uma letra w")
```

gerou um módulo único (-F) com 10 megabytes.

32.1.1 Passagem de parâmetros

Uma grande sacada da chamada standalone de um programa é passar parâmetros a ele na linha de chamada. No C, isto funciona como parâmetros da função `main()`. Eis como fica em Python

- Incluir o módulo `sys` no módulo (`import sys`)
- a variável `sys.argv[0]` contém o nome do programa
- o tamanho `len(sys.argv)` contém o número de parâmetros
- `sys.argv[1]` é o primeiro
- e assim por diante

Capítulo 33

Bibliografia

33.1 Python

para saber mais sobre Python

DEA20 DEITL e DEITL. **Intro to Python for Computer Science and Data Science**. USA, Pearson, 2020. 888p.

BIE20 BUELTA, Jaime. **Python Automation Cookbook**. Birmingham, Packt. 2020, 505p.

BEA13 BEAZLEY, David e JONES, Brian K. **Python Cookbook**. São Paulo, Novatec, 2013. 713p.

BUL20 BUELTA, Jaime. **Python Automation Cookbook**. Birmingham, Packt, 2020. 508p.

RAM15 RAMALHO, Luciano. **Python Fluente**. São Paulo, Novatec, 2015. 797p.

MEN10 MENEZES, Nilo Ney Coutinho. **Introdução à Programação com Python**. São Paulo, Novatec, 2010. 327p.

MCK18 MCKINNEY, Wes. **Python para Análise de Dados**. São Paulo, Novatec, 2018. 615p.

MUE16 MUELLER, John Paul, **Começando a programar em Python para leigos** Rio de Janeiro, Alta Books, 2016. 379p.

KIN15 KINSLEY, Harrison e McGUGAN, Will. **Introdução ao desenvolvimento de Jogos em Python com PyGame**. São Paulo, Novatec, 2015.

PYTxx Comunidade Python Mundial. Site <https://www.python.org>.

PYTxx Comunidade Python Brasil. Site <https://python.org.br>.

KAN18 KANTEK, Pedro Luis Garcia Navarro. **Python**. Curitiba, UFPR, 2018. Disponível em www.pkantek.com.br/ativ

DOW16 DOWNEY, Allen B. **Pense em Python**. São Paulo, Novatec, 2016.

FERsd FERG, Steven. **Pensando em Tkinter**. São Paulo, se, sd. Disponível em <http://thinkingtkinter.sourceforge.net>

33.2 Algoritmos e programação

Livros e materiais sobre programação.

GOO13 GOODRICH, M, TAMASSIA, R e GOLDWASSER, M. **Data Structure & Algorithmns in Python**. USA, John Wiley & Sons. 2013. 747p.

DEL12 DELGADO, Armando Luiz. **Linguagem C++**. Curitiba, UFPR, 2012.

DEW89 DEWDNEY, A. K. **The (new) Turing Omnibus**. New York, W. H. Freeman, 1989. 460p.

COR14 CORMEN, Thomas H. **Algoritmos**. Rio de Janeiro, Elsevier, 2014. 188p.

BHA17 BHARGAVA, Aditya. **Entendendo Algoritmos**. São Paulo, Novatec, 2017. 259p.

SKI97 SKIENA, Steven S. - **The Algorithm Design Manual**. New York, Springer. 1997. 486p.

SKI03 SKIENA, Steven S e REVILLA, Miguel A. **Programming Challenges**. New York, Springer. 2003. 361p.

-
- SKI12** SKIENA, Steven S e REVILLA, Miguel A. **Desafios de Programación**. New York, Springer. 2012. 343p. Tradução do inglês ao espanhol.
- FOR05** FORBELLONE, A. L. V. e EBERSPÄCHER, H. F. - **Lógica de Programação**. São Paulo. 2005. 214p.
- FOR72** FORSYTHE, Alexandra I. et all. **Ciência de Computadores - 1. curso**. Volume 1. Rio de Janeiro, Ao Livro Técnico. 1972. 234p.
- FOR72b** FORSYTHE, Alexandra I. et all. **Ciência de Computadores - 1. curso**. Volume 2. Rio de Janeiro, Ao Livro Técnico. 1972. 560p.
- KNU71** KNUTH, Donald E. **The Art of Computer Programming**. Volumes 1 e 2. Reading, Massachussets, Addison Wesley Publishing Company, 1971, 624p.
- WIR89** WIRTH, Niklaus. **Algoritmos e estruturas de dados**. Rio de Janeiro, Prentice-Hall do Brasil, 1989, 245p.
- KAO08** KAO, Ming-Yang. (ed.) **Encyclopedia of Algorithms**. New York, Springer, 2008. 1200p.

33.3 Métodos Numéricos

- KIU13** KIUSALAAS, Jaan. **Numerical Methods in Engineering with Python 3**. USA, Cambridge, 2013.
- SAN07** SANCHES, Ionildo José e FURLAN, Diogenes. **Métodos Numéricos**. Curitiba, UFPR, 2007.
- FRA07** FRANCO, Neide Bertoldi. **Cálculo Numérico**. São Paulo, Pearson, 2007.
- JUS18** JUSTO, Dagoberto Adriano et alli. **Cálculo Numérico, versão Python**. Porto Alegre, UFRGS, 2018.
- BAR87** BARROSO, Leônidas Conceição et alli. **Cálculo Numérico (com aplicações)**. São Paulo, Harbra, 1987.
- RUG88** RUGIERO, Márcia e LOPES, Vera Lúcia. **Cálculo Numérico - aspectos teóricos e computacionais**. São Paulo, Pearson, 1988.
- PAC78** PACITTI, Tércio e ATKINSON, Cyril. **Programação e Métodos Computacionais**. vol 1 e 2. Rio de Janeiro, LTC, 1979.
- CAM14** CAMPOS Filho, Frederico Ferreira. **Algoritmos Numéricos**. Rio de Janeiro, LTC, 2014.

O Symbolic Python é um pacote Python que implementa um sistema de computação simbólica (CAS=Computer Algebra System) no ambiente Python. Para entender o que é um sistema deste tipo, acompanhe:

```
comput. numérica: sqrt(8)=2.82842712
comput. simbólica: sqrt(8)=2*sqrt(2)
```

A vantagem da segunda é que é fácil passar da segunda para a primeira, mas não é possível o sentido oposto. Acompanhe:

```
>>> print(sp.sqrt(8))
2*sqrt(2)
>>> print(sp.sqrt(8).n())
2.82842712474619
>>> print(sp.sqrt(8).n(30))
2.82842712474619009760337744842
```

O paradigma de computação simbólica é o Maple. As vantagens de usar o sympy versus o Maple (por exemplo) são pelo menos duas: sympy é de graça (Maple custa 1.000 dólares) e por ser em python admite que você use tudo o que já sabe de python e também permite usar os mais de 135.000 pacotes também de graça.

Instalação e uso O sympy precisa ser instalado. Na versão para windows, você deve localizar o diretório scripts e nele fazer `pip install sympy` em um computador que tenha acesso à Internet. Se quiser usar winpython, nele o sympy já está pré-instalado. Num ou noutro, precisa importar o pacote. Faça `from sympy import *` ou algum comando equivalente. Este pacote exige como dependência o pacote NUMPY. A documentação completa do sympy ocupa hoje 2.127 páginas A4. Sua última versão deve ser buscada na Internet (procure `sympy doc`). Este é um produto relativamente recente na história de Python (2013), e portanto alguma instabilidade pode aparecer. Paciência, nem tudo nasce pronto neste planeta.

Começo

```
>>> import sympy as sp
>>> sp.init_session()
```

Importação Neste material, vai-se executar sempre o comando de importação

```
import sympy as sp
init_session() # estabelece ambiente
```

Iniciando Uma novidade neste pacote é o surgimento de variáveis simbólicas que terão um comportamento diferente das variáveis convencionais do Python. Quem cria variáveis simbólicas é o comando `symbols()`, veja no exemplo

```
>>> x,y,z = sp.symbols('x,y,z')
```

A partir deste ponto as variáveis `x`, `y` e `z` estarão disponíveis para armazenar expressões simbólicas. Uma definição alternativa, usando ferramentas do Python é

```
>>> a0,a1,a2,a3 = sp.symbols('a0:4')
```

Note que falando estritamente, não é obrigatório que os símbolos tenham o mesmo nome das variáveis, mas para evitar fortes emoções, vamos usar sempre o mesmo nome.

Expressões Pode-se definir expressões em Sympy usando símbolos junto com as operações básicas da matemática além de outras funções. Para escrever expressões, deve-se obedecer ao padrão Python (por exemplo $2x$ deve ser escrito como `2*x`, a potência x^2 como `x**2`). Há uma diferença importante em que a igualdade em sympy deve ser escrita como `Eq(a,b)` para

representar $a = b$. Ou melhor ainda, se $a = b$, então $a - b = 0$ e daí pode-se escrever em sympy `a-b`.

Há um método em sympy chamado `equals` que testa 2 expressões em pontos aleatórios e retorna True se todos resultarem iguais.

```
>>> a,b,x=symbols('a,b,x')
>>> a=cos(x)**2-sin(x)**2
>>> b=cos(2*x)
>>> a.equals(b)
True
```

Substituição A substituição é usada para avaliar uma equação num ponto

```
>>> expr
cos(x) + 1
>>> expr.subs(x,pi/2)
1
>>> (x**2+2*x+10).subs(x,3)
25
```

Também é usada para trocar uma subexpressão por outra, como em

```
>>> (x**2+2*x+10).subs(x,y**2)
y**4 + 2*y**2 + 10
```

Às vezes, o uso de métodos primitivos pode levar a resultados indesejados, como em

```
>>> expr=sin(2*x)+cos(2*x)
>>> expand_trig(expr)
2*sin(x)*cos(x) + 2*cos(x)**2 - 1
```

Ao quisermos a simplificação apenas do seno duplo, faz-se

```
expr.subs(sin(2*x),2*sin(x)*cos(x))
2*sin(x)*cos(x) + cos(2*x)
```

Note-se que o método `subs` devolve outra instância de expressão já que a expressão original permanece imutável. A substituição pode ser de mais de uma variável/expressão

```
>>> expr=3*x**3+4*y**2-2*z
>>> expr.subs([(x,3),(y,5),(z,1)])
179
```

Simplificação Não confundir com simplificação. Trata-se de transformar uma string em uma expressão sympy

```
>>> a="x**2+3"
>>> a
'x**2+3'
>>> b=sympify(a)
>>> b
x**2 + 3
>>> b.subs(x,1)
4
```

Impressão Há vários dispositivos em sympy. `pprint()` imprime a saída na tela. Há `latex` e `mathml`, entre outros.

```
>>> pprint(Integral(sqrt(1/x),x))
```

$$\int \sqrt{\frac{1}{x}} dx$$

```
>>> latex(Integral(sqrt(1/x),x))
'\\int \\sqrt{\\frac{1}{x}}\\, dx'
```

Simplificação Sympy tem dúzias de funções para fazer diversos tipos de simplificação. Há uma função principal chamada `simplify()` que aplica todas estas simplificações escolhendo depois a a mais simples. Isto nem sempre funciona bem, já que a expressão *a mais simples* não está claramente definida.

```
>>> simplify(sin(x)**2+cos(x)**2)
1
>>> simplify((x**3+x**2-x-1)/(x-1))
x**2 + 2*x + 1
>>> simplify(x**2+2*x+1)
x**2 + 2*x + 1
```

expand() e factor() Para este último caso, pode-se escolher `expand()` ou `factor()` que são dois simplificadores importantes em sympy.

```
>>> expand((x+1)**3)
x**3 + 3*x**2 + 3*x + 1
>>> factor(x**3+3*x**2+3*x+1)
(x + 1)**3
```

A expansão de um polinômio coloca-o na forma de uma soma de monômios. Já fatoraçãõ transforma a soma de monômios em uma multiplicação de fatores irredutíveis. Note-se que a entrada de ambos não precisa ser um polinômio estrito senso.

```
>>> init_printing()
>>> expand((cos(x)+sin(x))**2)
sin(x)**2 + 2*sin(x)*cos(x) + cos(x)**2
```

Outro simplificador é `collect()` que junta potências comuns dentro da expressão.

```
collect(x*y+x-3+2*x**2-z*x**2+x**3,x)
x**3 + x**2*(2 - z) + x*(y + 1) - 3
```

cancel() `cancel()` coloca qualquer função racional na forma canônica $\frac{p}{q}$ onde p e q são polinômios expandidos sem fator comum e com fatores inteiros.

```
>>> e=symbols('e')
>>> e=1/x+(3*x/2-2)/(x-4)
>>> cancel(e)
```

$$\frac{3.x^2 - 2.x - 8}{2.x^2 - 8.x}$$

apart() `apart()` executa uma decomposição de frações parciais de uma função racional.

```
>>> e=(x**3+x**2+1)/(x**4+x**2)
```

$$\frac{x}{x^2 + 1} + \frac{1}{x^2}$$

trigsimp() Simplifica usando identidades trigonométricas circulares e hiperbólicas.

```
>>> trigsimp(sinh(x)/tanh(x))
cosh(x)
```

expand_trig() Expande funções trigonométricas, ou seja, aplica as identidades de ângulos duplos ou soma de ângulos

```
>>> expand_trig(sin(x+y))
sin(x)*cos(y) + sin(y)*cos(x)
>>> expand_trig(tan(2*x))
```

$$\frac{2.tan(x)}{1 - tan(x)^2}$$

Potência As potências obedecem às seguintes identidades:

- $x^a x^b = x^{a+b}$
- $x^a y^a = (xy)^a$ só é verdade quando $x, y \geq 0$ e $a \in \mathbb{R}$
- $(x^a)^b = x^{ab}$ só é verdade quando $b \in \mathbb{Z}$

Veja-se para as regras 2 e 3: $(-1)^{1/2}(-1)^{1/2} \neq (-1. - 1)^{1/2}$ e $((-1)^2)^{1/2} \neq (-1)^{2.1/2}$ Esta observação é importante pois o sympy só aplica simplificações quando elas são sempre verdadeiras. Assim, as regras 2 e 3 só serão aplicadas se expressamente suas pré-condições forem explicitamente declaradas.

powsimp() Aplica as identidades 1 e 2 da esquerda para a direita

```
>>> powsimp(x**a*x**b)
a + x
b
```

expand_power_exp(), expand_power_base() e powdenest() Aplicam as identidades acima.

Logaritmos Têm tratamento similar às potências. Há 2 identidades:

1. $\log(xy) = \log(x) + \log(y)$
2. $\log(x^n) = n \cdot \log(x)$

Nenhuma dessas 2 é verdadeira para $x, y \in \mathbb{C}$. Para serem verdadeiras, $x, y \geq 0$ e $n \in \mathbb{R}$.

expand_log() Aplica as identidades 1 e 2 da esquerda para a direita.

```
>>> expand_log(log(x*y))
log(x*y)
>>> expand_log(log(x*y),force=True)
log(x) + log(y)
```

logcombine() aplica as identidades 1 e 2 da direita para a esquerda.

```
>>> logcombine(log(x)+log(y))
log(x) + log(y)
>>> logcombine(log(x)+log(y),
force=True)
log(x*y)
```

factorial(n) e binomial(n,k)

rewrite() Permite reescrever uma função em termos de outra

```
>>> tan(x).rewrite(cos)
```

$$\frac{\cos(x - \frac{\pi}{2})}{\cos(x)}$$

diff() Acha a derivada de uma função. Seu formato **diff(função, variável)**.

```
>>> diff(exp(x**2),x)
```

$$2x \cdot e^{x^2}$$

```
>>> diff(tan(x),x)
```

2

```
tan(x) + 1
```

Para conseguir derivadas múltiplas, passe a variável diversas vezes, ou a variável seguida da ordem da derivada.

```
>>> diff(x**4,x,x,x)
```

24.x

```
>>> diff(x**4,x,3)
```

24.x

Derivative() estabelece literalmente a derivada, tendo o mesmo formato de diff(). Para forçar o cálculo posterior, chame **doit()**.

```
>>> e=exp(x)+sin(x)
```

```
>>> e
```

$$e^x + \sin(x)$$

```
>>> diff(e,x)
```

$$e^x + \cos(x)$$

```
>>> Derivative(e)
```

$$\frac{d}{dx} e^x + \sin(x)$$

```
>>> Derivative(e).doit()
```

$$e^x + \cos(x)$$

integrate() Há 2 tipos de integrais: a definida e a indefinida. Para a integral indefinida é **integrate(função, variável)**.

```
>>> integrate(cos(x))
```

```
sin(x)
```

Note-se que não aparece a constante aditiva. Se ela for necessária deve ser manualmente incluída ou então o problema deve ser redesenhado para ser resolvido como uma equação diferencial.

Para computar uma integral definida, o formato é **integrate(func, (var, linf, lsup))**

```
>>> integrate(exp(x),(x,0,oo))
oo
>>> integrate(exp(-x),(x,0,oo))
1
>>> integrate(exp(-x**2-y**2),
(x,-oo,oo),(y,-oo,oo))
```

pi

Também há **Integral()** e depois para implementá-lo, chame **doit()**.

```
>>> e=log(x)**2
>>> Integral(e,x)
```

$$\int \log(x)^2 dx$$

```
>>> Integral(e,x).doit()
```

2

```
x*log(x) - 2*x*log(x) + 2*x
```

limit() Seja o limite $\lim_{x \rightarrow x_0} f(x)$

```
>>> limit(sin(x)/x,x,0)
```

```
1
```

$$\lim_{x \rightarrow 0^+} \frac{\cos(x) - 1}{x}$$

```
>>> Limit((cos(x)-1)/x,x,0).doit()
```

```
0
```

Note-se que limites à esquerda e a direita, exigem um quarto parâmetro que pode ser '+' ou '-' respectivamente.

series() Permite a expansão de séries em torno a um ponto. Para computar a expansão de $f(x)$ em torno do ponto $x = x_0$ em termos de x^n , usa-se **f(x).series(x,x0,n)**. x_0 e n podem ser omitidos, no caso em que serão assumidos $x_0 = 0$ e $n = 6$.

```
>>> exp(sin(x)).series(x,0,4)
```

$$1 + x + \frac{x^2}{2} + O(x^4)$$

solveset() A função principal para solução de equações. Sua sintaxe é **solveset(equação, variável, domínio)**. A equação $x = y$ deve ser escrita usando **Eq(x,y)** ou mais facilmente como **x-y**.

```
>>> solveset(x**2-x,x)
```

```
{0, 1}
```

```
>>> solveset(sin(x)-1,x,domain=Reals)
```

$$2.n.\pi + \frac{\pi}{2} / \in \mathbb{Z}$$

linsolve() Resolve sistemas de equações lineares. Seu formato **linsolve([eq1,eq2,...],[v1,v2,...])**.

Seja o sistema: $2x + 3y - z = 4$, $x + y + z = 7$ e $x - 2y + 3z = 9$

```
>>> linsolve([2*x+3*y-z-4,x+y+z-7,
x-2*y+3*z-9],[x,y,z])
```

```
{(1, 2, 4)}
```

```
>>> linsolve(Matrix([[2,3,-1,4],
[1,1,1,7],[1,-2,3,9]]),(x,y,z))
```

```
{(1, 2, 4)}
```

nonlinsolve() Resolve sistemas não lineares.

```
>>> nonlinsolve([x**2+y-z+1,2*x+y+z-8,
```

```
x-2*y+3*z-9],[x,y,z])
```

```
{(-1, 4, 6), (1, 2, 4)}
```

dsolve() Resolve equações diferenciais. Para tanto deve-se criar funções

```
>>> f,g=symbols('f,g',cls=Function)
```

```
>>> f(x)
```

```
f(x)
```

```
>>> f(x).diff(x)
```

```
d
```

```
--(f(x))
```

```
dx
```

Supondo a equação diferencial $f''(x) - 2f'(x) + f(x) = \sin(x)$ deve-se fazer

```
>>> dx=(f(x)).diff(x,x)-
2*f(x).diff(x)+f(x)-sin(x)
>>> dx
```

$$f(x) - \sin(x) - 2 \cdot \frac{d}{dx}(f(x)) + \frac{d^2}{dx^2}(f(x))$$

```
>>> dsolve(dx,f(x))
```

$$f(x) = (C_1 + C_2 \cdot x) \cdot e^x + \frac{\cos(x)}{2}$$

Matrizes Criação de matrizes podem ser feitas com **Matrix([[l1],[l2],...])**, com **eye(ordem)**, **zeros(lin,col)**, **ones(lin,col)** e **(diag(vetor))** em que o vetor é colocado na diagonal.

Matrizes podem ser operadas como em **M+N**, **3*M**, **M**2** e **M**-1** (esta última traz a matriz inversa).

```
>>> M=Matrix([[1,2,3],[1,2,1],[1,0,6]])
```

```
>>> M
```

```
[1 2 3]
```

```
[1 2 1]
```

```
[1 0 6]
```

```
>>> M**-1
```

```
[-3 3 1]
```

```
[5/4 -3/4 -1/2]
```

```
[1/2 -1/2 0]
```

det() calcula o determinante de uma matriz

```
>>> M.det()
```

```
-4
```

rref() Calcula a forma reduzida por linhas, retornando uma tupla de 2 elementos. O primeiro é a matriz reduzida por linhas e o segundo os índices das colunas pivot.

```
>>> N=Matrix([[2,3,-1,4],[1,1,1,7],
[1,-2,3,9]])
```

```
>>> N
```

```
[2 3 -1 4]
```

```
[1 1 1 7]
```

```
[1 -2 3 9]
```

```
>>> N.rref()
```

```
[1 0 0 1]
```

```
[0 1 0 2], (0, 1, 2))
```

```
[0 0 1 4]
```

Numpy

Este pacote implementa o conceito de array n-dimensional. Também o conceito de funções universais além de centenas de funções e constantes a usar em computação matemática e científica. É pré-requisito em muitos outros pacotes (ex: scipy, sympy, matplotlib, tensorflow, pygame, astropy, opencv, pulp, pandas entre outros).

O pacote é muito extenso e completo, seu manual tem 1542 páginas e pode ser encontrado via google, buscando [numpy-ref-1.12.0.pdf](#) ou uma versão mais recente.

Instalação Tendo acesso à Internet, entre no diretório de instalação do Python, no subdiretório **Scripts** dentro da árvore de instalação do Python. Abrindo uma janela CMD nesse diretório execute

```
pip install numpy
```

Se não souber fazer isto ou não tiver direitos na máquina em questão, mude de estratégia e use o Winpython. Para seu governo este pacote bem como muitos outros, já está pré-instalado no **winpython**.

Importação Há duas maneiras de importar. A mais simples é

```
>>> from numpy import *
```

 Aqui todos os objetos do pacote são incluídos no espaço de nomes atual. A segunda maneira, mais higiênica, é atribuir um nome ao pacote e referenciar seus objetos qualificando-os por este nome. O comando agora é

```
>>> import numpy as np.
```

 O nome do pacote agora é **np**. Pode-se usar qualquer outro nome, mas este material (e 99% da comunidade Python) o conhece como **np**.

Observar ainda que há diversos subpacotes dentro de numpy. Isso surgirá mais adiante em alguns exemplos. Os mais usados são **linalg** que agrega funções da álgebra linear, **fft** da transformada discreta de Fourier, **random** que trata diversas funções de geração de aleatórios e de distribuições estatísticas.

arranjos Um arranjo em numpy é um array n-dimensional. Quando n=1, o array é unidimensional logo é um vetor e neste caso é parecido com a lista convencional. A coisa começa a melhorar quando n=2 (matriz) ou mesmo quando n é maior do que 2. Em qualquer caso o a classe do pacote é ndarray e alguns de seus métodos são:

array Cria um array a partir de dados já existentes. Tipo tem muitas opções, sendo as mais comuns int, float, complex.

```
>>> import numpy as np
>>> a=np.array(elementos, tipo)
>>> b=np.array([1,2,3,4])
>>> b
array([1, 2, 3, 4])
>>> c=np.array([1,2,3],float)
>>> c
array([1., 2., 3.])
>>> d=np.array([[1,2,3],[0,0,7],
               [3,3,9]])
>>> d
array([[1, 2, 3],
       [0, 0, 7],
       [3, 3, 9]])
>>> d1=np.array(d,float)
>>> d1
array([[ 1.,  2.,  3.]])
```

```
[ 0.,  0.,  7.],
 [ 3.,  3.,  9.]])
```

Interessante que se você mandar imprimir a matriz ela será exibida sem as vírgulas

```
>>> print(d)
[[1 2 3]
 [0 0 7]
 [3 3 9]]
```

`np.set_printoptions(suppress=True)`
→ desabilita impressão em notação científica.

Na criação as listas podem ser fornecidas como tuplas (usando-se parênteses ao invés de colchetes), mas esta estratégia não será desenvolvida aqui.

Indexação Elementos individuais podem ser recuperados dos arrays. Podem também ser alterados. Cada dimensão do array exigirá uma especificação, separadas por vírgulas. Acompanhe

```
>>> d[1,2]
7
>>> c[1]
2.0
>>> d[1,1]=99
>>> d
array([[ 1,  2,  3],
       [ 0, 99,  7],
       [ 3,  3,  9]])
```

Fatiamento Usando a mesma estratégia do python, sub-arrays podem ser obtidos usando fatiamento

```
>>> e=np.array([[4,7,8,10],
               [11,12,16,19],[20,21,23,25],
               [40,41,50,51]])
>>> e
array([[ 4,  7,  8, 10],
       [11, 12, 16, 19],
       [20, 21, 23, 25],
       [40, 41, 50, 51]])
>>> e[1:3,1:3]
array([[12, 16],
       [21, 23]])
>>> e[0:4,1:4]
array([[ 7,  8, 10],
       [12, 16, 19],
       [21, 23, 25],
       [41, 50, 51]])
>>> e[1,1:4]
array([12, 16, 19])
```

Zeros Outra maneira de criar o array contendo zeros. O formato é `zeros((forma),tipo)`. Acompanhe:

```
>>> f=np.zeros((3,2),complex)
>>> f
array([[0.+0.j, 0.+0.j],
       [0.+0.j, 0.+0.j],
       [0.+0.j, 0.+0.j]])
>>> g=np.zeros((4,5),int)
>>> print(g)
[[0 0 0 0 0]
 ...
 [0 0 0 0 0]]
```

O **full** preenche com um valor pré-determinado.

```
>>> a = np.full((3,12),-1,int)
```

cria uma matriz de 3 linhas por 12 colunas de inteiros iguais a -1.

Outros criadores O **eye** funciona como a matriz identidade, mas valendo para matrizes não quadradas. A identidade exige apenas a ordem da matriz (já que ela é quadrada).

```
>>> h=np.eye(3,2)
>>> h
array([[1., 0.],
       [0., 1.],
       [0., 0.]])
>>> j=np.identity(3)
>>> j
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

Atributos Um array tem os atributos **ndim** que é um escalar indicando a quantidade de dimensões e **shape** que é uma tupla indicando a quantidade de elementos em cada dimensão.

```
>>> j.ndim
2
>>> j.shape
(3, 3)
>>> g.shape
(4, 5)
```

Reorganizar Para reorganizar um array já existente, usa-se o método **reshape**.

```
>>> a1=np.arange(0,15,2)
>>> a1
array([0,2,4,6,8,10,12,14])
>>> a2=a1.reshape((2,4))
>>> a2
array([[ 0,  2,  4,  6],
       [ 8, 10, 12, 14]])
```

Para reorganizar in-place usa-se o método **resize**.

```
>>> b1=np.array([1,7,8,9,10,11])
>>> b1.resize([3,2])
>>> print(b1)
[[ 1  7]
 [ 8  9]
 [10 11]]
```

Funções a seguir algumas (tem inúmeras mais, veja a documentação):

```
>>> a=np.array([[1,4,6],[7,8,1]])
>>> print(a)
[[1 4 6]
 [7 8 1]]
```

a.mean()	média
a.var()	variância
a.sum()	soma
a.prod()	produto
a.max()	maior valor
a.min()	menor valor
a.argmax()	índice do maior
a.argmin()	índice do menor
np.unique(a)	valores únicos

```
>>> print(a.mean())
4.5
>>> print(a.max())
8
>>> print(a.argmax())
4
>>> print(np.unique(a))
[1 4 6 7 8]
```

Funções Universais As funções universais (ufuncs) se aplicam elemento a elemento dentro dos arrays. Acompanhe o exemplo abaixo para identificar o funcionamento das ufuncs.

```
>>> c1=[1,2,3]
>>> c2=[3,7,9]
>>> c1+c2
[1, 2, 3, 3, 7, 9]
>>> d1=np.array([1,2,3])
>>> d2=np.array([3,7,9])
>>> d1+d2
array([ 4, 9, 12])
```

Broadcasting Para broadcasting (aplicar um escalar a um array)

```
>>> a é array([1,2,3,4])
>>> a+3
array([4,5,6,7])
>>> b*5
array([0.5, 3.5, 40., 45.])
```

multiplicação matricial

```
>>> a1=np.array([[1,2,-4],
                 [3,-1,5]])
>>> a2=np.array([[6,-3],
                 [1,-2],[2,4]])
>>> np.dot(a1,a2)
array([[ 0, -23],
        [27, 13]])
```

Produto vetorial

```
>>> x = (1, 2, 0)
>>> y = (4, 5, 6)
>>> print np.cross(x, y)
[12 -6 -3]
```

Determinante

```
>>> m=np.array([[2,3],[-1,-2]])
>>> m
[[ 2 3]
 [-1 -2]]
>>> np.linalg.det(m)
-1.0
```

Matriz inversa

```
>>> good = np.array([[2.1,3.2],
                    [4.3,5.4]])
>>> np.linalg.inv(good)
[[-2.23140496  1.32231405]
 [ 1.7768595  -0.8677686 ]]
```

Sistema de equações lineares A solução de um sistema de equações lineares por exemplo

```
2x + y = 19
x - 2y = 2
cuja solução é (x=8 and y=3):
>>> coef=np.array([[2,1],[1,-2]])
>>> cont=np.array([19, 2])
>>> np.linalg.solve(coef, cont)
[ 8. 3.]
```

Se $m > n$, onde m são equações e n são incógnitas, o comando é `c=np.linalg.lstsq(a,b)` que devolve a solução que minimiza o erro cometido. (`solve`, acima, exige $m = n$).

Transposta

```
>>> a=np.array(range(6),
                float).reshape((2,3))
>>> a é array([[ 0., 1., 2.],
              [ 3., 4., 5.]])
>>> a.transpose()
array([[0.,3.],[1.,4.],[2.,5.]])
```

Concatenação

```
>>> a=np.array([1,2],float)
>>> b=np.array([3,4,5],float)
>>> c=np.array([7,8,9],float)
>>> np.concatenate((a,b,c))
array([1.,2.,3.,4.,5.,7.,8.,9.])
>>> a=np.array([[1,2],[3,4]])
>>> b=np.array([[5,6],[7,8]])
>>> np.concatenate((a,b))
array([[ 1., 2.],
        [ 3., 4.],
        [ 5., 6.],
        [ 7., 8.]])
>>> np.concatenate((a,b), axis=0)
array([[ 1., 2.],
        [ 3., 4.],
        [ 5., 6.],
        [ 7., 8.]])
>>> np.concatenate((a,b), axis=1)
array([[ 1., 2., 5., 6.],
        [ 3., 4., 7., 8.]])
```

Funções matemáticas abs, sign, sqrt, log, log10, exp, sin, cos, tan, arcsin, arccos, arctan, sinh, cosh, tanh, arcsinh, arccosh, and arctanh. floor (inteiro inferior), ceil (inteiro superior), e rint (inteiro mais próximo = arredondado).

Geração de aleatórios Gera números reais entre 0 e 1.

```
>>> np.random.rand(2,2)
array([[0.07724462, 0.93519669],
        [0.44908731, 0.12275935]])
```

Inteiros aleatórios. O formato é início (inclusive), final (exclusive) e quantidade. Os dois últimos são opcionais.

```
>>> np.random.randint(1,50,3)
array([40, 4, 49])
>>> np.random.randint(6) # dado
4
```

Ordenação

```
>>> a=np.array([[1,4,7],[8,3,9],
                [22,11,13]])
>>> a
array([[ 1, 4, 7],
        [ 8, 3, 9],
        [22, 11, 13]])
>>> np.sort(a)
array([[ 1, 4, 7],
        [ 3, 8, 9],
        [11, 13, 22]])
>>> np.sort(a,axis=None)
array([1,3,4,7,8,9,11,13,22])
>>> np.sort(a,axis=0)
array([[ 1, 3, 7],
        [ 8, 4, 9],
        [22, 11, 13]])
```

Distribuição normal Esta função retorna simulações de uma distribuição normal. Para obter $N(\mu, \sigma^2)$, basta fazer `sigma*np.random.randn(...)+mu`. Onde μ = média da distribuição e σ^2 é a variância. Veja-se o exemplo

```
>>> 2.5 * np.random.randn(2, 4) + 3
array([[ -4.49,  4.00, -1.81,  7.29],
        [ 0.39,  4.68,  4.99,  4.84]])
```

Cria um array 2,4 contendo simulações de 3,6.25. Lembre que $2.5^2 = 6.25$.

Distribuição binomial Cria simulações desta distribuição. Usa 2 parâmetros: n , que significa o número de tentativas e p que vem a ser a probabilidade de sucesso da distribuição ($0 \leq p \leq 1$). A função retorna para cada simulação o número de sucessos. Lembrando que a probabilidade é

$$P(N) = \binom{n}{N} p^N (1-p)^{n-N}$$

Veja-se o exemplo:

```
>>> n=10
>>> p=0.5
>>> s=np.random.binomial(n,p,1000)
```

O exemplo acima faz 1000 simulações, cada uma jogando 10 moedas e informando a quantidade de caras em cada simulação.

Desvio padrão em arrays

```
>>> a = np.array([[1, 2], [3, 4]])
>>> np.std(a)
1.1180339887498949
>>> np.std(a, axis=0)
array([ 1., 1.])
>>> np.std(a, axis=1)
array([ 0.5, 0.5])
```

Embaralhando uma lista Supondo uma lista L já existente e querendo embaralhá-la, no próprio local

```
>>> L=[1,4,7,22,11,5,4,77]
>>> L
[1, 4, 7, 22, 11, 5, 4, 77]
>>> np.random.shuffle(L)
>>> L
[4, 5, 7, 1, 4, 22, 77, 11]
```

Ler e Gravar dados Suponha uma variável x como em

```
>>> x=np.random.rand(3,2)
>>> x
array([[0.65615773, 0.64436913],
        [0.54172899, 0.9726057 ],
        [0.27936652, 0.09892232]])
```

Para guardá-la em disco (note que este formato é ilegível por humanos) para posterior recuperação pode-se fazer

```
>>> x.dump('c:/apl2/lixo.dad')
```

E depois basta fazer

```
>>> y=np.load('c:/apl2/lixo.dad')
```

Para colocar uma constante Imagine um array de 6 linhas por 10 colunas que deve ser preenchido com 22

```
>>> z=np.empty((10,6))
>>> z.fill(22)
```

Iteração com elementos

```
>>> a=np.arange(9).reshape(3,3)
>>> a
array([[0, 1, 2],
        [3, 4, 5],
        [6, 7, 8]])
>>> for x in np.nditer(a):
...     print(x)
...
0 1 2 3 4 5 6 7 8
```

Para medir alguma execução

```
import time
ini=time.time()
... o que quer medir
fim=time.time()
print("tempo em segs",fim-ini)
```

Matplotlib

Pacote para fazer desenhos e manipulações de imagens em Python. Sua documentação é muito extensa, o manual do usuário tem 2303 páginas e sua última versão deve ser buscada na internet procurando [matplotlib.pdf](#).

Instalação Tendo acesso à Internet, entre no diretório de instalação do Python, no subdiretório `Scripts` dentro da árvore de instalação do Python. Abrindo uma janela CMD nesse diretório execute

```
pip install matplotlib
```

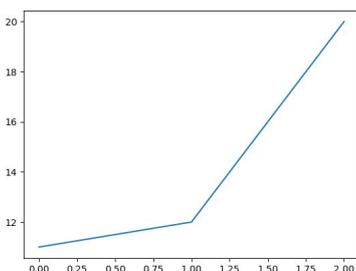
Se não souber fazer isto ou não tiver direitos na máquina em questão, mude de estratégia e use o Winpython. Este pacote bem como muitos outros, já está pré-instalado no winpython.

Importação Há duas maneiras de importar. A mais simples é `>>> from matplotlib import *` Aqui todos os objetos do pacote são incluídos no espaço de nomes atual. A segunda maneira, mais higiênica, é atribuir um nome ao pacote e referenciar seus objetos qualificando-os por este nome. O comando agora é `>>> import matplotlib.pyplot as plt`. O nome do pacote agora é `plt`. Pode-se usar qualquer outro nome, mas este material (e 99% da comunidade Python) o conhece como `plt`. Não esqueça de importar também o pacote numérico (`import numpy as np`).

Modos de uso Há 2 modos de uso do pacote: usar a interface procedural `pyplot` ou usar a API nativa da matplotlib, esta mais orientada a objetos. A primeira é constituída por métodos (como `xlabel`, `legend`, etc) e elas podem ser usadas de duas maneiras: se chamadas sem argumentos, devolvem o valor atual do parâmetro e se se chamadas com parâmetros elas atualizam este valor.

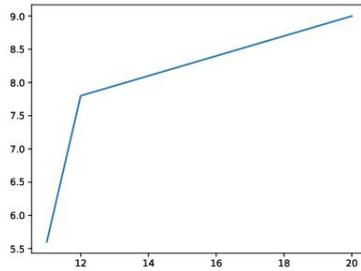
Plot Cria gráficos `xy`. Crie as variáveis a plotar. Se for uma única série, ela será considerada como `y` (e neste caso `x` serão seus índices).

```
import numpy as np
import matplotlib.pyplot as plt
a=[11,12,20]
lixo=plt.plot(a)
plt.show()
```



Ou se quiser crie um conjunto de valores `x`, `y` como dois vetores.

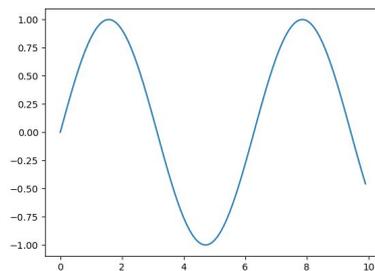
```
a=[11,12,20]
b=[5.6, 7.8, 9.0]
lixo=plt.plot(a,b)
```



Diversas curvas podem ser desenhadas no mesmo gráfico, bastando chamar o plot com uma matriz (cada coluna será uma curva) ou chamando várias vezes a `plot()` uma vez para cada coluna e vendo tudo de uma vez com `plt.show()`.

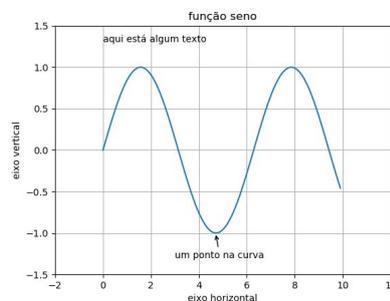
Usando uma função

```
x = np.arange(0.,10.,0.1)
x
array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. , 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, ... , 9.9])
y = np.sin(x)
ll = plt.plot(x,y)
plt.show()
```



Melhorando o visual

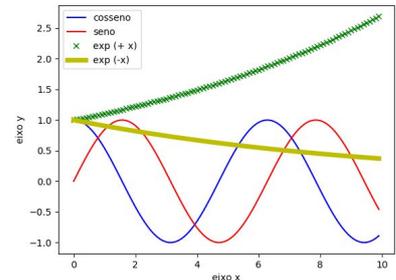
```
xl=plt.xlabel('eixo horizontal')
yl=plt.ylabel('eixo vertical')
ttl=plt.title('função seno')
ax=plt.axis([-2,12,-1.5,1.5])
grd=plt.grid(True)
txt=plt.text(0,1.3,'aqui está algum texto')
ann=plt.annotate('um ponto na curva',
xy=(4.7,-1),xytext=(3, -1.3),
arrowprops=dict(arrowstyle='->'))
```



Mais

```
x=np.arange(0.,10,0.1)
a=np.cos(x)
b=np.sin(x)
c=np.exp(x/10)
d=np.exp(-x/10)
la=plt.plot(x,a,'b-',label='cosseno')
lb=plt.plot(x,b,'r-',label='seno')
lc=plt.plot(x,c,'gx',label='exp(+x)')
```

```
ld=plt.plot(x,d,'y-',linewidth=5,
label='exp(-x)')
ll=plt.legend(loc='upper left')
lx=plt.xlabel('eixo x')
ly=plt.ylabel('eixo y')
plt.show()
```

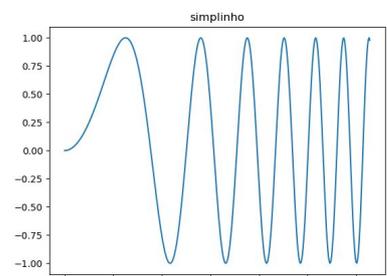


Usando uma função própria Defina a função como em

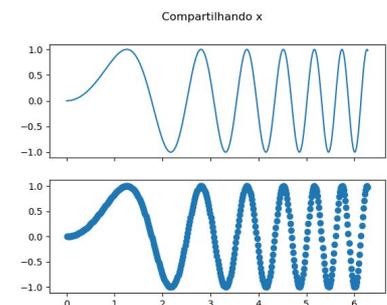
```
def f(x):
return x**2/10+x**3/100
t1=np.arange(0, 10, 0.1)
plt.plot(t1, f(t1), 'r-')
plt.show()
```

subplots Permite colocar diversos gráficos no mesmo desenho. O formato da chamada é `subplots(lin,col,sharex=T/F,sharey=T/F)`. PS: `scatter` é para desenhar o ponto sozinho, sem ligação. Acompanhe

```
x=np.linspace(0, 2*np.pi, 400)
y=np.sin(x**2)
fig,ax=plt.subplots()
ax.plot(x,y)
ax.set_title('simplinho')
plt.show()
```



```
...
f,axarr=plt.subplots(2, sharex=True)
adj = plt.subplots_adjust(hspace=0.4,wspace=0.4) #
f.suptitle('Compartilhando x')
axarr[0].plot(x,y)
axarr[1].scatter(x,y)
plt.show()
```



Obviamente se o conjunto de subplots for (2,2), a indexação sera [0,0], depois [0,1] depois [1,0] e assim por diante.

Tortas

Eis o formato do comando `pie`

```
matplotlib.pyplot.pie(x, explode=None,
labels=None, colors=None, autopct=None,
pct-distance=0.6, shadow=False,
labeldistance=1.1, startangle=None,
radius=None, counterclock=True,
wedgeprops=None, textprops=None,
center=(0, 0), frame=False,
rotatelabels=False, hold=None, data=None)
```

As fatias estão dadas por x . Cada fatia ocupa $x/\text{sum}(x)$. Se a soma de x é menor do que 1, os valores de x indicam a fração e não são normalizados. As fatias são desenhadas em sentido anti-horário a partir do eixo x .

explode O parâmetro `explode` se não for `None` será um array de tamanho `len(x)` indicando qual a fração do raio que cada fatia deve ser deslocada.

labels é uma sequência de strings indicando o que é cada fatia.

colors é uma sequência de cores. Se usado `None` usa-se a sequência padrão.

autopct Pode ser `None`, um string ou uma função. Diz para identificar a fatia com um número. Se for uma função ela será chamada. A `pctdistance` indica onde por este rótulo.

shadow é um booleano, indica se haverá sombra.

labeldistance o default é 1.1 e indica a distância radial onde os labels vão ser desenhados.

startangle onde começa a primeira fatia (a partir do eixo x).

radius O raio da pizza. Se `None` vale 1.

counterclock Booleano, indica o sentido anti-horário ou não.

wedgeprops é um dicionário (como em `wedgeprops = {'linewidth': 3}`). Para saber mais veja a lista de argumentos do objeto `wedge`.

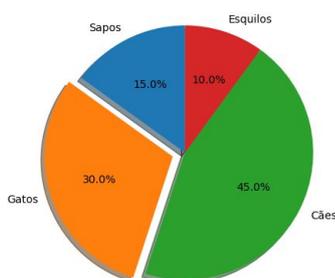
textprops é um dicionário com atributos de texto.

center a localização do centro da pizza (default 0,0)

frame Booleano. Se true desenha um frame.

rotatelabels Booleano: rotaciona cada fatia no ângulo da fatia.

```
labels='Sapos','Gatos','Cães','Esquilos'
sizes = [15, 30, 45, 10]
explode = (0, 0.1, 0, 0)
fig1, ax1 = plt.subplots()
ax1.pie(sizes,explode=explode,
labels=labels,autopct='%1.1f%%',
shadow=True,startangle=90)
ax1.axis('equal')
plt.show()
```

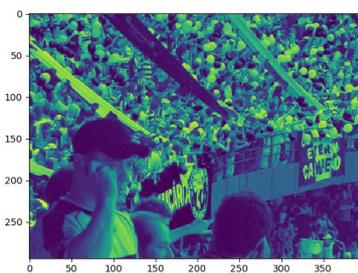


Brincando com imagens

```
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.image as mpimg
img=mpimg.imread('c:/python/coxa.png')
#print(img.shape) é (294, 400, 3)
imgplot=plt.imshow(img)
plt.show(imgplot)
```



```
lum_img=img[:, :, 0]
imgplot=plt.imshow(lum_img)
plt.show(imgplot)
```

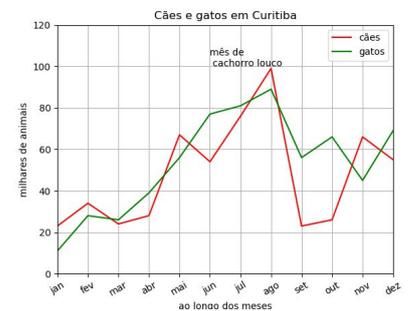


```
im22=plt.imshow(lum_img, cmap='hot')
plt.show(im22)
```



Exemplo Completo

```
# caogato
import matplotlib.pyplot as plt
import numpy as np
def caogato():
ca=[23,34,24,28,67,54,76,99,23,26,66,55]
ga=[11,28,26,39,56,77,81,89,56,66,45,69]
plt.plot(ca, 'r-', label='cães')
plt.plot(ga, 'g-', label='gatos')
plt.legend()
plt.title('Cães e gatos em Curitiba')
plt.xlabel('ao longo dos meses')
plt.ylabel('milhares de animais')
plt.axis([0,11,0,120])
plt.xticks(np.arange(12), ('jan', 'fev',
'mar', 'abr', 'mai', 'jun', 'jul', 'ago',
'set', 'out', 'nov', 'dez'), rotation=30)
plt.text(5,100, 'mês de\n cachorro louco')
plt.grid()
plt.savefig('c:/python/caogato.png')
plt.show()
caogato()
```



Diferença entre subplot e subplots

```
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(1234567)
data=np.random.randn(2,200)
#array 2L x 200C # dp=1 e media 0
fig,ax=plt.subplots(2,2)
ax[0,0].hist(data[0])
ax[0,0].set(ylabel='eixo y')
ax[1,0].scatter(data[0],data[1])
ax[0,1].plot(data[0],data[1])
ax[1,1].hist2d(data[0],data[1])
plt.show()
```

```
fig2=plt.figure(1)
sp=plt.subplot(221)
l1=plt.hist(data[0])
ly=plt.ylabel("eixo y")
sp=plt.subplot(222)
l1=plt.scatter(data[0],data[1])
sp=plt.subplot(223)
l1=plt.plot(data[0],data[1])
sp=plt.subplot(224)
l1=plt.hist2d(data[0],data[1])
plt.show()
```

Python

É a linguagem da hora porque:

- * é moderna, usa e abusa de recursos de máquina.
- * é freeware
- * tem 170.000 pacotes, todos free também
- * usa bignumers, usa matemática complexa, ...
- * multiplataforma (mesmo)
- * fácil de aprender e usar
- * Unicode e UTF-8 por padrão
- * está no mainstream. Consulte www.tiobe.com

Pode ter certeza que usando Python, você estará em boa companhia

Instalação

Localize seu sistema operacional e capacidade de endereçamento do seu sistema (32 ou 64 bits) e daí vá em www.python.org escolha e instale a versão mais indicada. Sempre a 3, nunca a 2 que vai sair de linha.

Tipos básicos

int	123, -45, ...
float	12.3, -4.5, -1.5e-3...
bool	True ou False
str	'ola', "Curitiba", "let's go"

Os inteiros podem ser binários **0b010**, octais **0o654** ou hexadecimais **0xF5**. Dentro de strings, podem haver caracteres de escape como em "aqui \n vamos". Neste caso \n pula de linha.

Identificadores

nomes para variáveis, funções, módulos, classes,...

a..zA..Z seguido de a..zA..0..9_

- diacríticos (caracteres acentuados) podem ser usados, mas não devem
- palavras reservadas proibidas
- minúsculas ≠ maiúsculas

Use nomes que façam sentido para você.

Assinalamento de variáveis

Usa **a=b**. A expressão **b** é avaliada e seu valor é associado ao nome **a**.

<code>x=1+6.1+cos(z)</code>	
<code>a=b=c=0</code>	as 3 variáveis recebem zero.
<code>a,b,c=3,5,7.8</code>	a recebe 3, b 5 e c 7.8.
<code>a,b=b,a</code>	a e b trocam o valor
<code>a,*b=lista</code>	Separa lista em cabeça e cauda
<code>a+=1</code>	Incrementa a em 1
<code>del xy</code>	Remove a variável xy

Matemática

Operações: +, -, *, ** (potência), / (divisão real), // (divisão inteira), % (resto). 10/3 é 3.3333333 enquanto 10//3 é 3. Pode ser que se exija a importação do módulo **math**.

adição	<code>a+b</code>	<code>2+3 → 5</code>
subtração	<code>a-b</code>	<code>2-3 → -1</code>
multiplicação	<code>a*b</code>	<code>3*5 → 15</code>
divisão real	<code>a/b</code>	<code>10/3 → 3.33333</code>
div. inteira	<code>a//b</code>	<code>10//3 → 3</code>
potência	<code>a**b</code>	<code>2**3 → 8</code>
resto	<code>a%b</code>	<code>10%3 → 1</code>
módulo	<code>abs(a)</code>	<code>abs(-3) → 3</code>

seno	<code>sin(a)</code>	<code>sin(pi/4) → 0.707...</code>
cosseno	<code>cos(a)</code>	<code>cos(2*pi/3) → 0.49</code>
tangente	<code>tan(a)</code>	<code>tan(1) → 1.55...</code>
\sqrt{a}	<code>sqrt(a)</code>	<code>sqrt(2) → 1.414...</code>
$\log_e(a)$	<code>log(a)</code>	<code>log(e**2) → 2</code>
teto	<code>ceil(a)</code>	<code>ceil(3.1) → 4</code>
chão	<code>floor(a)</code>	<code>floor(3.6) → 3</code>

Entrada e Saída

```
aa=int(input('Forneça o valor: '))
```

O programa pára, imprime a mensagem **Forneça o valor:** aguarda o usuário digitar algo e este valor, devidamente transformado em **inteiro** é armazenado na variável **aa**. A resposta com comando **input** SEMPRE volta no formato **string**, pelo que a conversão exata deve ser claramente estabelecida.

```
a=1;b=5
print('x=',a,' c=',b)
# vai imprimir x= 1 c= 5
```

Tipos de contenedores

lista: sequência ordenada, acesso por índice, não necessariamente homogênea. Reconhecida pelos [, ,]. Exemplo [1,4,9], ['alfa',3,0.99], ...

dicionário: tabela de chave:conteúdo, acesso por chave. Reconhecida por {chave:conteudo, ,}. Exemplo {1:"paraná", 2:"piauí", ...}.

Conjunto: dados não ordenados, sem duplicações. Acessados por operações de conjuntos. Reconhecido por {} ou set(). Exemplo {1,3,77,90}

tuplas: listas fixas, que não podem sofrer alteração. Reconhecidas por (, ,). Exemplo (1,2,3).

Conversões

<code>int("123")</code>	o valor 123
<code>int("2F",16)</code>	38
<code>int(12.3)</code>	12
<code>float(1)</code>	1.0
<code>float(-3.23e4)</code>	-32300.0
<code>round(1.7)</code>	2
<code>chr(64)</code>	@
<code>ord('@')</code>	64
<code>bytes([52,65,67])</code>	b'4AC'
<code>list("IJK")</code>	['I', 'J', 'K']

```
dict([(1,"um"),(2,'dois')])
→ {1:'um',2:'dois'}
set('um','do') → {'um','do'}
'*.join(['um','do']) → 'um*do'
'um dois 3'.split()
→ ['1', 'dois', '3', '4']
[int(x) for x in ('1','33','267')]
→ [1, 33, 267]
```

Lógica Booleana

Comparadores em Python: ==, !=, >=, <=, >, <. Conectores: and, or e not. Constantes booleanas: True e False.

Fatiamentos

Para listas, tuplas, strings, bytes,... mas não para dicionários e conjuntos.

a=	[10, 20, 30, 40, 50]
índs neg	-5 -4 -3 -2 -1
índs pos	0 1 2 3 4
fatia pos	0 1 2 3 4 5
fatia neg	-5 -4 -3 -2 -1

```
len(a) → 5
a[0] → 10
a[-1] → 50
a[1] → 20
```

Agora, sempre no formato [início:até fim:increm.]

```
a[:-1] → [10, 20, 30, 40]
a[1:-1] → [20, 30, 40]
a[::2] → [10, 30, 50]
a[::-1] → [50, 40, 30, 20, 10]
a[::2] → [50, 30, 10]
a[:] → [10, 20, 30, 40, 50]
(cópia rasa da seq)
a[1:3] → [20, 30]
a[-3:-1] → [30, 40]
a[3:] → [10, 20, 30]
a[3:] → [40, 50]
```

Importação de módulos

No comando **import aaa**, o Python vai buscar o módulo **aaa.py**. Note que o S.O. precisa achar o arquivo (alterar PATH, por exemplo).

Em **import aaa**, o elemento de nome **bbb** do módulo vai ser acessado escrevendo **aaa.bbb**.

Em **import aaa as xxx**, o elemento de nome **bbb** do módulo vai ser acessado escrevendo **xxx.bbb**.

Em **from aaa import ***, o elemento de nome **bbb** do módulo vai ser acessado escrevendo **bbb**.

Condicional

```
if condição:
    executado se cond==True
```

O bloco indentado só é executado se a condição for verdadeira.

```
if condição:           Exemplo:
bloco-1                if x<6:
else:                   print('a')
    bloco-2            else:
bloco-3                 print('b')
                        print('c')
```

O bloco1 é executado se a condição é True. Se a condição é False, executa-se o bloco2. Em qualquer um dos casos, bloco3 é executado incondicionalmente. No exemplo, se **x = 5** imprimem-se as letras **a** e **c**.

Há ainda o comando **elif**, veja

```
if a<=100:
    bloco-1
elif a<=500:
    bloco-2
elif a<=1000:
    bloco-3
else:
    bloco-4
continuação
```

Se **a < 100**, executa-se o bloco1 e depois a continuação. Se **100 < a ≤ 500** executa-se o bloco2 e depois a continuação. Se **500 < a ≤ 1000** executa-se o bloco3 e depois a continuação. Se **a > 1000** executa-se o bloco 4 e depois a continuação.

Repetição

```
while condição:
    bloco-1
bloco-2
```

O bloco1 será executado repetidas vezes, **enquanto** a condição for verdadeira. Ao final o bloco2 será normalmente executado.

Saída de repetição

Existem 2 saídas mandatórias para laços `while` e `for`. São: `break` que determina a saída imediata do laço. Há também `continue` que força o reexame da condição de saída.

Iteradores

Os iteradores estão em python como os iteradores estão em C, mas são muito mais abrangentes. Além de ser elementos em um vetor, podem ser elementos em uma lista, tupla, dicionário, conjunto ou registro em um arquivo, entre outros. Exemplos:

```
a={1:300,5:400,7:500}
for i in a: # impressos:
    print(i) # 1, 5 e 7
a=(1,20,300)
for i in a: # impressos:
    print(i) # 1, 20 e 300
a={1,6,8}
for i in a: # impressos:
    print(i) # 1, 6 e 8
a=[1, 6, 8, 9, 11, 20]
print([x for x in a if x%2==0])
# impressos [6,8,20]
[x**2 for x in range(5)]
# imprime [0, 1, 4, 9, 16]
[(x, x**2) for x in range(6)]
# [(0,0),(1,1),(2,4),(3,9),(4,16),(5,25)]
[str(round(pi,i)) for i in range(1,5)]
# ['3.1', '3.14', '3.142', '3.1416']
```

Operações em contenedores

As operações matemáticas (sum, min,...) exigem dados numéricos.

tamanho	<code>len(c)</code>	<code>len([1,2,3])</code> → 3
mínimo	<code>min(c)</code>	<code>min([3,2,7])</code> → 2
máximo	<code>max(c)</code>	<code>max([3,2,7])</code> → 7
soma	<code>sum(c)</code>	<code>sum([1,2,3])</code> → 6
sort	<code>sort(c)</code>	<code>sort([3,2,7])</code> → [2,3,7]
pertença	<code>x in c</code>	3 in [3,2,7] → True
todos	<code>all(c)</code>	verdadeiro se todos os <i>c</i> são verdadeiros
algum	<code>any(c)</code>	verdadeiro se algum <i>c</i> é verdadeiro
reverso	<code>reversed(c)</code>	<i>c</i> em ordem reversa
posição	<code>c.index(v)</code>	índice de <i>v</i> em <i>c</i>
contador	<code>c.count(v)</code>	quantas ocorrências de <i>v</i> em <i>c</i>

Operações em listas

inserção	<code>L.append(v)</code>	insere <i>v</i> ao final de <i>L</i>
inserção múltipla	<code>L.extend(p)</code>	insere a lista <i>p</i> no final de <i>L</i>
inserção	<code>L.insert(i,v)</code>	insere <i>v</i> após o índice <i>i</i> em <i>L</i>
remoção	<code>L.remove(v)</code>	remove a 1ª ocorrência de <i>v</i> em <i>L</i>
pop	<code>L.pop([i])</code> → <i>v</i>	remove e retorna <i>L[i]</i>
ordena	<code>L.sort()</code>	ordena <i>in place</i>
reverte	<code>L.reverse()</code>	reverte <i>in place</i>

Operações em dicionários

chave	<code>d[chave]=v</code>	atualiza a chave de <i>d</i> com <i>v</i>
chave	<code>d[chave]</code> → <i>v</i>	consulta a chave de <i>d</i>
limpa	<code>D.clear()</code>	limpa o dicionário <i>D</i>
exclui	<code>del D[chave]</code>	elimina a entrada <i>chave</i>

atualiza	<code>D.update(d2)</code>	atualiza a entrada <i>d2</i>
chaves	<code>D.keys()</code>	chaves interadas
valores	<code>D.values()</code>	valores interadas
pares	<code>D.items()</code>	pares <i>chave,valor</i> interadas

Para saber se alguém (chave) existe no dicionário: `a=d.get(chave)`. Se *a* não existir, ela vale None.

Operações em conjuntos

união		conjunto união
intersecção	&	conjunto intersecção
diferença	-	$a=\{1,4,5,6,8,9\}$, $b=\{2,3,4,5\}$, $a-b \rightarrow \{8,1,6,9\}$
diferença simétrica	^	$a^b \rightarrow \{1,2,3,6,8,9\}$
adição	<code>S.add(v)</code>	Adiciona elemento <i>v</i> no dicionário <i>S</i>
exclusão	<code>S.remove(v)</code>	remove <i>v</i> no dicionário <i>S</i>
limpeza	<code>S.clear()</code>	zera o dicionário <i>S</i>

Arquivos

abertura	<code>f=open(a,t)</code>	abre o arquivo <i>a</i> e cria a variável <i>f</i> . <i>t</i> pode ser <i>w</i> , <i>r</i> ou <i>a</i>
leitura	<code>a=f.read(n)</code>	Lê <i>n</i> caracteres de <i>f</i> e joga em <i>a</i>
escrita	<code>f.write(q)</code>	grava <i>q</i> no arquivo <i>f</i>
le linha	<code>f.readline()</code>	lê a próxima linha em <i>f</i>
le linhas	<code>f.readlines(n)</code>	lê <i>n</i> próximas linhas em <i>f</i>
fechamento	<code>f.close()</code>	Fecha o arquivo <i>f</i>
aplica	<code>f.flush()</code>	aplica mudanças em <i>f</i>

```
def arq():
    f=open('c:/apl2/vivoblo1',"r")
    for linha in f:
        print(linha)
    arq()
```

Sequências inteiras

`range([inicio,]fim[,etapa])`

inicio se omitido, vale 0. *etapa* se omitido, vale 1. *fim* encerra antes deste valor.

<code>range(5)</code>	→ [0,1,2,3,4]
<code>range(3,8)</code>	→ [3,4,5,6,7]
<code>range(2,12,3)</code>	→ [2,5,8,11]
<code>range(20,5,-5)</code>	→ [20,15,10]

Definição de funções

```
def numefun([parâmetros]):
    """documentação"""
    ...computações...
    return resultado
```

Os parâmetros e todas as variáveis definidas na função só existem enquanto a função é executada. São portanto locais. O comando `return` encerra a execução da função. Comandos após ele nunca serão executados. Após a definição, a função pode ser chamada, como em

```
aa = numefun(x,y,z)
```

Seu resultado será armazenado em *aa* e as variáveis *x*, *y*, *z* serão usadas no lugar dos parâmetros usados lá na definição. Os parênteses indicam que é uma função.

Operações em strings

<code>S.split([e])</code>	Cria lista, separando pelo <i>e</i> [espaços]
<code>[j].join(L)</code>	junta elementos de <i>L</i> , conectando-os com <i>j</i>
<code>S.strip([c])</code>	Remove <i>c</i> ou brancos no início e fim de <i>S</i>
<code>S.count('c')</code>	Conta quantos <i>c</i> há em <i>S</i>
<code>S.partition(e)</code>	Separa em 3 pedaços: antes, <i>e</i> e depois
<code>S.find(u,i,f)</code>	Retorna o primeiro <i>u</i> começando em <i>i</i> , terminando em <i>f</i> ou -1
<code>S.is...()</code>	... pode ser <code>alpha...</code> Retorna True se sim
<code>S.upper()</code>	Retorna maiúsculas
<code>S.lower()</code>	Retorna minúsculas
<code>S.title()</code>	Só as iniciais maiúsculas
<code>S.swapcase()</code>	troca a caixa
<code>S.center(t,[f])</code>	centraliza em <i>t</i> preenchendo com <i>f</i>
<code>S.startswith(p,i,f)</code>	True se <i>S</i> começa com <i>p</i>
<code>S.endswith(u,i,f)</code>	True se <i>S</i> termina com <i>u</i>

Formatação

"padrão".`format(x,y,z...)` → string
padrão é formado por diversas {}, cada uma podendo ter: seleção:formato!conversão. seleção é a sequência de variáveis. Se omitida, vale a mesma ordem em que aparecem. formato pode ter: caráter de preenchimento alinhamento sinal tamanho-mínimo tamanho-máximo tipo. tipo pode ser `bcdoXeEfgGs%`. Conversão pode ser `sr`. Eis alguns exemplos

```
from datetime import *
print('{} {}'.format(1,2)) # 1b2
print('{} {}'.format(1,2.1)) # 1b2.1
print('{1} {0}'.format(1,2)) # 2bb1
print('{:f}'.format(3.14159265))
# 3.141593
print('{:d}'.format(2)) # 2
print('{:5d}'.format(2)) # bbb2
print('{:06.2f}'.format(3.1415))
# 003.14
print('{:%Y-%m-%d %H:%M}'.format(
    datetime(2019, 2, 3, 4, 5)))
# 2019-02-03 04:05
```

A partir do Python 3.6 através da PEP 498, surgiram os formatted string literals, conhecidos como f-strings, que simplificaram e uniformizaram a formatação de dados numéricos. O formato básico é

```
f' texto... {var:[alin][tam].decs f}...'
```

Objetos/Classes

Objetos são a soma de estrutura de dados + métodos. São perfeitos para descrever o universo e encapsular a complexidade.

```
class Carro:
    def __init__(self, marca, ano):
        #construtor
        self.marca=marca
        self.ano=ano
    def metodo(self):
        print("a marca e ",self.marca)
class Colecao(Carro):
    # colecao herda Carro
    def __init__(self,m,a,c):
        Carro.__init__(self,m,a)
        self.cor=c
c1=Carro("fusca",1970)
c1.metodo()
c2=Colecao("chevete",1973,"branco")
```