Listas

- Criada usando colchetes, por exemplo:

  `A = [1, 3, 77, 8, 90]`
- Ao contrário de outras linguagens (C, C++, Java, PHP, APL, ...) admite elementos heterogêneos.

  `A = [1, 'ola', 3, 77, 8, 90, "agora vai"]`
- Pode ser vazia (`A=[]`).
- Acesso para leitura `X=A[5]`, como para atualização `A[5]=33`.

Indexação

- Índice positivo: da esquerda para a direita
- Em uma lista com n elementos e índices positivos:
  - O primeiro é o 0 (zero)
  - O último é o $n - 1$
- Índice negativo: da direita para a esquerda
- Em uma lista com n elementos e índices negativos:
  - O primeiro é o $-n$
  - O último é o -1

Vejam-se alguns exemplos

```
LA   =   [  8,    10,   'ola',   1290,    -4.5,     'aaa',    78]
i>=0        0      1       2        3        4         5        6
i<0        -7     -6      -5       -4       -3        -2       -1
```

Fatiamento

- Operação muito comum em Python (aparentemente copiada de MatLab)
- Tem como formato:

  `[ inicio : fim : incremento]`
- Tudo pode ser omitido. Quando incremento é omitido, pode-se omitir o segundo `:`
- início é omitido: assume-se o primeiro
- fim é otimitido: assume-se o último
- incremento é omitido: assume-se 1

Exemplos

```
>>> LB=[5, 17, 33, 80, 91]
>>> LB[:-1]
[5, 17, 33, 80]
>>> LB[1:-1]
[17, 33, 80]
>>> LB[::2]
[5, 33, 91]
>>> LB[::-1]
[91, 80, 33, 17, 5]
>>> LB[::-2]
[91, 33, 5]
>>> LB[1:3]
[17, 33]
>>> LB[-3:-1]
[33, 80]
>>> LB[:3]
```

```
>>> LB[3:]
[80, 91]
>>> LB[:]
[5, 17, 33, 80, 91]
```

Usando uma faca

- marque colchetes e vírgulas
- positivos: da esquerda para a direita, começando em zero
- negativos: da direita para a esquerda, começando em espaço e depois -1
- Corte nos riscos

```
LB = [   5,  17,  33,  80, 91 ]
      |    |    |    |    |    |
      0    1    2    3    4    5
     -5   -4   -3   -2   -1
```

Outras operações

```
>>> A=[1, 2]
>>> A.append(5)
>>> A
[1,  2,  5]
>>> A.insert(1,80)   #antes do 1
>>> A
[1,  80,  2,  5]
>>> A.remove(2)  #remove o primeiro
[1,  80,  5]
>>> A.pop()  # remove e retorna o último
>>> A.sort   # ordena a lista (homogênea)
>>> A.reverse() # reverte a lista
>>> len(A)  # retorna o tamanho da lista
>>> min(A), max(A), sum(A) # só para numéricas
```

Exercícios

1. Considere WH=[2, 3, 5, 8, 10, 12]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.

a) WH[3:4:3]
b) WH[-4:4:2]
c) WH[3:4:3]
d) WH[1:6:2]
e) WH[:5:2]

8 5 8 23 17

Exercícios

2. Considere LN=[2, 11, 16, 17, 22, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.

a) LN[1:4:2]
b) LN[3:6:2]
c) LN[1:4:2]
d) LN[-5:5]
e) LN[3:-2]

28 43 28 66 17

Exercícios

3. Considere UX=[11, 12, 14, 18, 21, 22]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.

a) UX[3:4:2]
b) UX[:4:2]
c) UX[3:4:2]
d) UX[2:-2]
e) UX[1:-1:2]

18 25 18 32 30

Exercícios

4. Considere HY=[4, 9, 14, 15, 18, 21, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.

a) HY[-7:7:2]
b) HY[:5:-1]
c) HY[:5:2]
d) HY[:5:-2]
e) HY[-7:-2:2]

64 28 36 28 36

Conjuntos

Um conjunto é uma lista de coisas (um conjunto ou como se diz em Python: *set*), com 2 características e algumas operações específicas. As características:

- Não importa a ordem original dos elementos
- Não há elementos repetidos: se houver o Python deixa uma cópia só.

Em razão da primeira propriedade os conjuntos não suportam indexação nem fatiamento.
O conjunto pode ser criado oferecendo um conjunto de elementos à função *set* ou mais simplesmente, colocando os elementos entre chaves.

Operações

```
>>> a = {1,2,3,4,5,3,2}
>>> a
{1, 2, 3, 4, 5}
>>> b=set([1,2,3,4,5,1])
>>> b
{1, 2, 3, 4, 5}
```

As operações associadas a conjuntos, são aquelas da matemática básica e são:

Intersecção: Dados dois conjuntos A e B, a intersecção de A e B, $A \cap B$ é o conjunto dos elementos que estão em A e em B. A intersecção usa o símbolo &. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A & B    é    {8, 6}
```

Operações

União dados A e B, conjuntos, a união de A com B, denotada $A \cup B$ é o conjunto dos elementos que estão em A ou em B. Seu símbolo é |.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A|B
{2, 4, 5, 6, 7, 8, 9, 10}
```

diferença Retorna os elementos de A que não estão em B. Seu símbolo -.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A-B    é   {2, 10, 4}
```

diferença simétrica Retorna todos os elementos de ambos os conjuntos que pertencem somente a um deles. O símbolo é ^. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
```

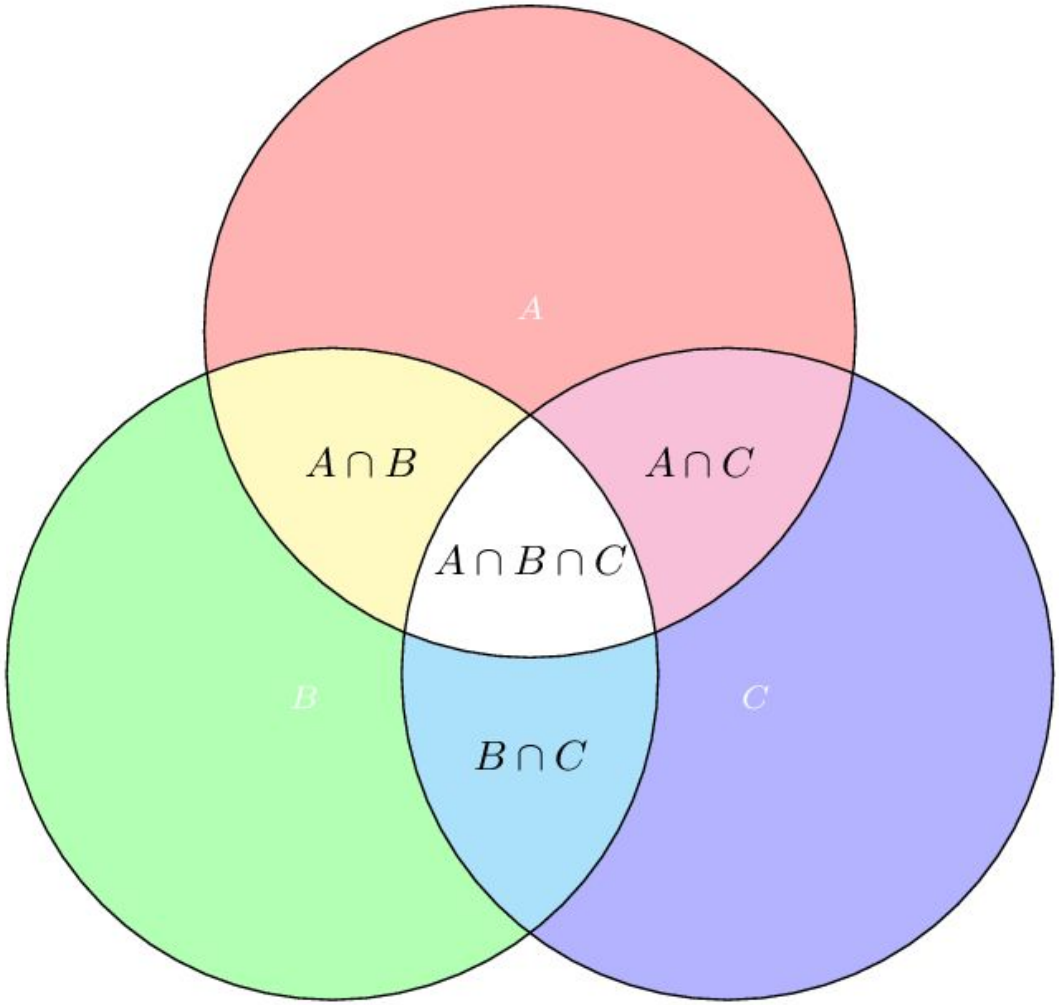
```
>>> A^B
{2, 4, 5, 7, 9, 10}
```

Exemplos

```
>>> a=[1,2,34,5,6,7]
>>> b=set([1,2,3,4,5,6])
>>> a
[1, 2, 34, 5, 6, 7]
>>> b
{1, 2, 3, 4, 5, 6}
>>> a=set(a)
>>> a
{1, 2, 34, 5, 6, 7}
>>> b
{1, 2, 3, 4, 5, 6}
>>> a|b          # uniao
{1, 2, 34, 3, 5, 6, 7, 4}
>>> a&b          # intersecçao
{1, 2, 5, 6}
```

```
>>> a-b  #diferença
{34, 7}
>>> a^b  # dif.simétrica
{34, 3, 4,7}
```

Exercício de conjuntos



Figuras em

c:/p/n/934/respostas.pdf

Exercício de conjuntos

Você deve achar a cardinalidade dos conjuntos a seguir descritos em 3 casos e deve somar os valores encontrados.

M= $A \cap B \cap C$

N= $A \cap B \cap C^c$

P= $A \cap B^c$

Q= $C \cap (A \cup B)^c$

R= $A \cap (B \cup C)$

S= $(B \cap (A \cup C)^c) \cup (A \cap C \cap (A \cap B \cap C)^c)$

T= $A \cup B \cup C$

U= $(A \cap B \cap C^c) \cup (A \cap C \cap B^c) \cup (B \cap C \cap A^c)$

V= $(A \cap B) \cup (A \cap C) \cup (B \cap C)$

W= $(A \cup B \cup C) \cap ((A \cap B) \cup (A \cap C) \cup (B \cap C))^c$

Lembre que $A \cap B^c \Leftrightarrow A - B$

Exercício de conjuntos

Lembre que $A \cap B^c \Leftrightarrow A - B$

Eis os valores a usar:

1	A	10	13	4	1	15	3	11	9
	B	6	3	7	10	5	15	9	11
	C	9	4	14	6	8	2	1	11
2	A	11	15	2	10	5	9	13	1
	B	14	7	13	9	10	1	4	5
	C	7	2	11	10	9	4	12	15
3	A	6	14	12	15	1	8	7	13
	B	3	9	8	2	15	4	12	13
	C	6	8	3	14	15	9	1	10

Para a resposta, preencha a cardinalidade (o número de elementos) de cada conjunto.

ex.	M	N	P	Q	R	S	T	U	V	W
1										
2										
3										
$\sum$										

Resposta

6 8 10 5 22 13 39 21 27 12

Dicionários

Uma estrutura de dados parecido a lista, mas com o detalhe: enquanto listas sao acessadas pelo índice, dicionários são acessados por uma parte de seu conteúdo, denominada chave. Assim, o dicionário é composto de pares: chave e conteúdo e o que o dicionário faz é associar uns a outros. Os dicionários são criados usando-se chaves `{}` O formato é

```
a = { chave: valor, chave: valor, ... }
```

`a` é o nome do diconário, e dentro dele há pares de chave:valor, separadas pelo caractere dois pontos `:`.

Mais ...

Depois de criado um dicionário é consultado escrevendo-se seu nome, colchete, a chave desejada e fecha-colchete. Acompanhe

```
>>> documentos={'rg':1234, 'motorista':34567,
'passaporte': 8998, 'coxa': 23454}
>>> documentos['passaporte']
8998
```

Quando se atribui um valor a uma chave, se a mesma não existir ainda, será criada com este valor. Se ela já existisse, teria seu valor alterado para aquele agora fornecido.
Um detalhe importante é que a ordem dos elementos dentro de um dicionário não pode ser importante pois não é garantida.
Se um acesso é feito a um dicionário para uma chave inexistente ocorre um erro de `Keyerror`. Para evitar tal erro, se necessário, deve-se fazer uma consulta com o operador `in`

```
if 'pis/pasep' not in doc:          doc['pis/pasep']=55
```

Mais ...

As chaves e os conteúdos podem ser acessados como se fossem uma lista

```
>>> documentos.keys()
dict_keys(['rg', 'motorista', 'passaporte',
   'coxa', 'pis/pasep'])
>>> documentos.values()
dict_values([1234, 34567, 8998, 23454, 555])
```

Bote que eles voltam na forma de geradores. Podem ser usados diretamente em ciclos `for`, ou transformados em lista usando-se `list`. Para apagar uma chave, usa-se a instrução `del`. Veja

```
>>> del documentos['coxa']
>>> documentos
{'rg': 1234, 'motorista': 34567,
   'passaporte': 8998, 'pis/pasep': 555}
```

Uma aplicação prática

Uma aplicação prática do uso de dicionários. Seja a sequência de Fibonacci que é $1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...$ e cuja formulação é:

$$\begin{cases} F(1) & = 1 \\ F(2) & = 1 \\ F(n) & = F(n-1) + F(n-2) \quad \textit{se } n \geq 3 \end{cases}$$

Implementando diretamente esta definição em Python, fica-se com

```
def fibo(n):
  if n<3:
    return 1
  else:
    return fibo(n-1)+fibo(n-2)
xx = int(input('informe n '))
print(fibo(xx))
```

Tempos

Cujos tempos de execução são: F(20)=6 milésimos de segundo; F(30)=0.42 segundos, F(40)=49 segundos, F(41)=80 segundos e F(42)= 131 segundos e paramos por aqui. O problema desta implementação é o recálculo do mesmo valor para a função muitas vezes. Acompanhe: para calcular fibo(3) fazem-se 2 chamadas a fibo (fibo(1) e fibo(2)). Para calcular fibo(4) fazem-se 2 chamadas a fibo(2) e fibo(3). Como fibo(3) são 2 chamadas, no total são 4 chamadas. Fibo(5) são 6, fibo(6) são 10, fibo(7), 16; fibo(8) 26, fibo(9) 42... e as chamadas também compõe uma outra sequência de Fibonacci. O caso é que para calcular f(100) o computador deve demorar mais de 1 mês de processamento. A alternativa é o que se chama *algoritmo memoizável* que nada mais faz do que guardar o resultado já calculado. Onde ? Neste caso em um dicionário.

Tempos

```
d = {1:1, 2:1}
def fibm(n):
  aa=d.get(n)
  if aa is None:
    d[n]=fibm(n-1)+fibm(n-2)
    return d[n]
  else:
    return aa
print(fibm(xx))
```

Olhe os tempos: fibm(30) demora 5 milésimos, fibm(40), 9 milésimos e fibm(100) demora 9 milésimos também.