

Funções em Python Além das centenas de funções pré-existentes em Python (print, min, max, sum, len, int, float, input...) a grande vantagem do ambiente é que você pode definir as funções que precisar para o seu trabalho. Depois que uma função foi definida ela passa a se comportar como se fosse (e é) uma função primitiva do Python.

Para definir uma nova função, usa-se o comando `def`, veja:

```
def nome(parâmetros):  
    comandos
```

Após a palavra `def` deve vir o nome (único) pelo qual a função será conhecida. Não pode ser uma palavra reservada do Python. Logo depois do nome, é obrigatório um "abre parênteses". A seguir uma quantidade opcional de parâmetros, que são como se fossem nomes de variáveis. Depois desses parâmetros vem o fecha parênteses. Se não houver parâmetro, depois do nome da função vem abre e fecha parênteses. Termina o cabeçalho o caráter "dois pontos" (:). Ele inicia um bloco que limita a definição da função.

Diferentemente do que aconteceu até agora, as linhas que fazem parte da função, não são executadas agora. Elas ficam armazenadas e serão executadas apenas quando a função agora definida, for chamada (executada).

A razão para definir uma função pode ser:

- * Para repetir o mesmo processamento em diversos locais de um programa.
- * Para encapsular um processamento específico.
- * Para comerciar ou apenas trocar algoritmos.
- * Para modularizar um grande processamento, permitindo que o mesmo seja desenvolvido por diversas pessoas.
- * Para poder manusear a complexidade crescente.

A função pode retornar algum resultado para quem a chama. Por exemplo, a função `len(lista)` retorna um valor que corresponde ao comprimento da lista que é passada como parâmetro.

Veja um exemplo de tudo que se disse

```
def maior(a):  
    i=len(a)-1 #1  
    mai=-99999 #2  
    while i>= 0: #3  
        if a[i]>mai: #4  
            mai=a[i] #5  
            i=i-1 #6  
    return mai #7  
    pass #8  
    #9
```

```
print(maior([1,5,7,2,3,17,5]))
```

Na linha #1 define-se a função de nome `maior` e se diz que ela deverá ser chamada com um único parâmetro. Quem fez a função sabe que este parâmetro único deve ser uma lista formada por números. Na linha #2 cria-se a variável `i` e atribui-se a ela o valor do tamanho da lista parâmetro menos 1. Como se sabe tal índice acessa o último elemento da lista. Na linha #3 define-se uma outra variável de controle apta a guardar o maior valor da lista até aquele momento. Como ainda nenhum item foi processado ela vale $-\infty$. Na linha #4 define-se que `i` vai variar até ter um valor negativo. Na linha #5 se o elemento de ordem `i` na lista for maior do que a variável `mai`, `mai` será atualizada com este novo valor. Na linha #7 que já está fora da indentação do `if`, mas ainda dentro da indentação do `while` o índice `i` é recuado uma posição e graças ao `while` há um retorno à linha #4. Finalmente, quando o ciclo `while` acabar, vai-se para a linha #8 que simplesmente devolve o valor calculado a quem chamou a função `maior` e o processamento desta função está encerrado. A linha #9 não é executada (de qualquer maneira `pass` não faz nada mesmo), mas lembre-se QUALQUER coisa colocada após um `return` é meramente decorativo, pois nunca vai ser executado. Na linha #10, já fora da função `maior` (note a indentação) a mesma vai ser executada. Veja que preciso colocar um `print` para externalizar o resultado da função. Se não houver `print`, a função vai ser executada, mas nada será visto de fora.

O parâmetro é uma lista ? Se o parâmetro for uma lista de 4 números, ele será um único parâmetro. Estruturalmente é diferente de 4 parâmetros escalares, que no fundo são uma lista de 4 números. Veja

```
def f1(L4):  
    print(L4[0], L4[1], L4[2], L4[3])  
# ----- OU -----  
def f2(a,b,c,d):  
    print(a, b, c, d)  
f1([1,2,3,4])  
f2(1,2,3,4)
```

Qual estratégia usar ao modularizar um problema e definir quantas e quais funções devem ser construídas é uma arte: a bússola nesta tarefa é a melhor maneira de reaproveitar código e de evitar a reinvenção da roada.

Variáveis locais e globais Como regra geral todas as variáveis definidas dentro de uma função (como `i` e `maior` no exemplo aí acima), são chamadas **locais** à função e isto significa que elas nunca sobrevivem ao tempo de execução da função: não existem antes da função e deixam de existir quando a função acaba. Desta maneira não podem ter seu valor consultado e/ou alterado por alguém externo à função.

Este fato permite que duas funções distintas tenham variáveis com o mesmo nome (como é muito comum um índice chamado `i`) sem que um atrapalhe o outro. Desta maneira a única forma de ceder e receber valores entre funções é através de parâmetros e de retornos.

Já uma variável que precise ser vista e alterada por outros é conhecida como variável **GLOBAL**. Ela deve ser definida dentro de um módulo, mas **FORA** de uma função. Agora ela pode ser manuseada por todas as funções desse módulo e por todos os módulos que importam o módulo que a definiu.

Para que uma função possa alterar uma variável global, a mesma deve ser definida como global usando-se o comando `GLOBAL` como linha seguinte à definição.

Veja um exemplo:

```
xu = 77  
def fun():  
    xu = 2  
    print(xu)  
print(xu) # 77 é impresso  
fun() # 2 é impresso  
print(xu) # 77 é impresso  
def alva():  
    global xu  
    xu=2  
alva()  
print(xu) # 2 é impresso
```

As boas práticas de programação sugerem não usar variáveis globais. Esta prática viola o encapsulamento e é fonte frequente de bugs de programação.

Seus módulos podem ser importados Organizando seu trabalho direitinho, você pode modularizar seus programas e importá-los quando necessário, acompanhe

```
#----- beta.py -----  
def alfa(x):  
    return x**3  
#-----  
>>> import beta  
>>> print(beta.alfa(4))  
#-----  
>>> from beta import *  
>>> print(alfa(4))
```

☞ Para você fazer

A seguir uma lista de funções que você deve programar em Python. Imprima as funções e devolva-as impressas junto com esta folha. Não esqueça de identificar com seu nome e o código 1 limite entrega 10/10/24

1. Em uma certa província a velocidade máxima permitida em estradas é de 100 km por hora e a velocidade mínima é de 50 km por hora. A multa por violar estes limites é de R\$ 10 para cada km por hora acima da velocidade máxima ou abaixo da velocidade mínima. Escreva uma função que receba a velocidade de condução e indique quanto

deve ser a multa se é que ela deva existir. Em caso negativo, retornar 0. Por exemplo, se for chamada com 75 km/h, deve retornar 0. Se for chamada com 110 ou com 40 Km/h deve retornar 100,00.

2. Escreva uma função que receba 2 horários (no formato HH, MM) indicando o início e o fim de uma determinada tarefa. A função deve devolver a duração em minutos da tarefa. Se a entrada for 1,10 e 2,0 a saída deverá ser 50.
3. Escreva uma função que receba uma lista de números e imprima-a em ordem reversa (de trás para a frente). Se a entrada for 4, 7, -3, 22, 14, a impressão deverá ser de 14, 22, -3, 7 e 4.
4. Escreva uma função que receba um parâmetro numérico e retorne a palavra **PAR** ou a palavra **IMPAR** se o parâmetro for par ou ímpar. Por exemplo, se a entrada for 3, a saída deverá ser **IMPAR**.
5. Escreva uma função que receba o valor de x e calcule e devolva o valor de $S = 1/2 + 2/3 + 3/4 + \dots + x/x + 1$. Se a entrada for 5 a saída deverá ser 3.55.
6. Escreva uma função que receba uma lista de números calcule e devolva a média dos elementos da lista. Se a entrada for 8 7 4 19 21 a saída deverá ser 11.8.
7. Escreva uma função que receba um número inteiro e devolva a soma de todos os divisores deste número. Se a entrada for 15 a saída deverá ser 24.
8. Escreva uma função que receba uma lista formada por 3 números (se vierem menos ou mais deve dar um erro e encerrar) e deve imprimir **SIM** se os números forem lados de um triângulo retângulo e **NÃO** se não puderem ser. Por exemplo, se a entrada for 3,5,4 a saída deverá ser **SIM**. Se a entrada for 1,10,100 a saída deverá ser **NÃO**.
9. Escreva uma função que receba uma lista de números e devolva a diferença entre o maior e o menor deles. Se a entrada for 33, 22, -10, -4 e 33 a saída deve ser 43.
10. Escreva uma função que receba dois números e devolva **True** se tais números puderem indicar uma hora e minuto e **False** senão. Por exemplo, se a entrada for 23,59 a saída deverá ser **True**, idem se for 0,0. Se a entrada for 33,5 a saída deverá ser **False**.



408-75789 - e /nt

Parâmetros opcionais Seja uma função para sublinhar algo. Poderia ser

```
def sublinha():  
    print("_"*30)
```

Entretanto, nesta função tanto o caracter sublinha (`_`) quanto o valor 30 são fixos. Para permitir alterar isso, pode-se fazer:

```
def sublinha2(n=30, caractere="_"):  
    print(caractere*n)
```

Veja agora como a coisa é chamada

```
>>> def sublinha(n=30, cara="_"):  
    print(cara*n)  
>>> sublinha()
```

```
-----  
>>> sublinha(n=10)
```

```
-----  
>>> sublinha(cara='$!')  
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
```

Funções em Python Além das centenas de funções pré-existentes em Python (print, min, max, sum, len, int, float, input...) a grande vantagem do ambiente é que você pode definir as funções que precisar para o seu trabalho. Depois que uma função foi definida ela passa a se comportar como se fosse (e é) uma função primitiva do Python.

Para definir uma nova função, usa-se o comando def, veja:

```
def nome(parâmetros):  
    comandos
```

Após a palavra def deve vir o nome (único) pelo qual a função será conhecida. Não pode ser uma palavra reservada do Python. Logo depois do nome, é obrigatório um "abre parênteses". A seguir uma quantidade opcional de parâmetros, que são como se fossem nomes de variáveis. Depois desses parâmetros vem o fecha parênteses. Se não houver parâmetro, depois do nome da função vem abre e fecha parênteses. Termina o cabeçalho o caráter "dois pontos" (:). Ele inicia um bloco que limita a definição da função.

Diferentemente do que aconteceu até agora, as linhas que fazem parte da função, não são executadas agora. Elas ficam armazenadas e serão executadas apenas quando a função agora definida, for chamada (executada).

A razão para definir uma função pode ser:

- * Para repetir o mesmo processamento em diversos locais de um programa.
- * Para encapsular um processamento específico.
- * Para comerciar ou apenas trocar algoritmos.
- * Para modularizar um grande processamento, permitindo que o mesmo seja desenvolvido por diversas pessoas.
- * Para poder manusear a complexidade crescente.

A função pode retornar algum resultado para quem a chama. Por exemplo, a função len(lista) retorna um valor que corresponde ao comprimento da lista que é passada como parâmetro.

Veja um exemplo de tudo que se disse

```
def maior(a):  
    #1          #1  
    i=len(a)-1  #2  
    mai=-99999  #3  
    while i>= 0:  
        #4  
        if a[i]>mai:  
            #5  
            mai=a[i] #6  
            i=i-1    #7  
    return mai    #8  
    pass         #9  
print(maior([1,5,7,2,3,17,5])) #10
```

Na linha #1 define-se a função de nome maior e se diz que ela deverá ser chamada com um único parâmetro. Quem fez a função sabe que este parâmetro único deve ser uma lista formada por números. Na linha #2 cria-se a variável i e atribui-se a ela o valor do tamanho da lista parâmetro menos 1. Como se sabe tal índice acessa o último elemento da lista. Na linha #3 define-se uma outra variável de controle apta a guardar o maior valor da lista até aquele momento. Como ainda nenhum item foi processado ela vale $-\infty$. Na linha #4 define-se que i vai variar até ter um valor negativo. Na linha #5 se o elemento de ordem i na lista for maior do que a variável mai, mai será atualizada com este novo valor. Na linha #7 que já está fora da indentação do if, mas ainda dentro da indentação do while o índice i é recuado uma posição e graças ao while há um retorno à linha #4. Finalmente, quando o ciclo while acabar, vai-se para a linha #8 que simplesmente devolve o valor calculado a quem chamou a função maior e o processamento desta função está encerrado. A linha #9 não é executada (de qualquer maneira pass não faz nada mesmo), mas lembre-se QUALQUER coisa colocada após um return é meramente decorativo, pois nunca vai ser executado. Na linha #10, já fora da função maior (note a indentação) a mesma vai ser executada. Veja que preciso colocar um print para externalizar o resultado da função. Se não houver print, a função vai ser executada, mas nada será visto de fora.

O parâmetro é uma lista ? Se o parâmetro for uma lista de 4 números, ele será um único parâmetro. Estruturalmente é diferente de 4 parâmetros escalares, que no fundo são uma lista de 4 números. Veja

```
def f1(L4):  
    print(L4[0], L4[1], L4[2], L4[3])  
# ----- OU -----  
def f2(a,b,c,d):  
    print(a, b, c, d)  
f1([1,2,3,4])  
f2(1,2,3,4)
```

Qual estratégia usar ao modularizar um problema e definir quantas e quais funções devem ser construídas é uma arte: a bússola nesta tarefa é a melhor maneira de reaproveitar código e de evitar a reinvenção da roda.

Variáveis locais e globais Como regra geral todas as variáveis definidas dentro de uma função (como i e maior no exemplo aí acima), são chamadas locais à função e isto significa que elas nunca sobrevivem ao tempo de execução da função: não existem antes da função e deixam de existir quando a função acaba. Desta maneira não podem ter seu valor consultado e/ou alterado por alguém externo à função.

Este fato permite que duas funções distintas tenham variáveis com o mesmo nome (como é muito comum um índice chamado i) sem que um atrapalhe o outro. Desta maneira a única forma de ceder e receber valores entre funções é através de parâmetros e de retornos.

Já uma variável que precise ser vista e alterada por outros é conhecida como variável GLOBAL. Ela deve ser definida dentro de um módulo, mas FORA de uma função. Agora ela pode ser manuseada por todas as funções desse módulo e por todos os módulos que importam o módulo que a definiu.

Para que uma função possa alterar uma variável global, a mesma deve ser definida como global usando-se o comando GLOBAL como linha seguinte à definição.

Veja um exemplo:

```
xu = 77  
def fun():  
    xu = 2  
    print(xu)  
print(xu) # 77 é impresso  
fun()     # 2 é impresso  
print(xu) # 77 é impresso  
def alva():  
    global xu  
    xu=2  
alva()  
print(xu) # 2 é impresso
```

As boas práticas de programação sugerem não usar variáveis globais. Esta prática viola o encapsulamento e é fonte frequente de bugs de programação.

Seus módulos podem ser importados

Organizando seu trabalho direitinho, você pode modularizar seus programas e importá-los quando necessário, acompanhe

```
#----- beta.py -----  
def alfa(x):  
    return x**3  
#-----  
>>> import beta  
>>> print(beta.alfa(4))  
#-----  
>>> from beta import *  
>>> print(alfa(4))
```

☞ Para você fazer

A seguir uma lista de funções que você deve programar em Python. Imprima as funções e devolva-as impressas junto com esta folha. Não esqueça de identificar com seu nome e o código 2 limite entrega 10/10/24

1. Escreva uma função que receba uma lista de números positivos e negativos misturados e devolva duas listas: a primeira contendo apenas os positivos e a segunda apenas os negativos, na mesma ordem em que eles apareceram na entrada. Se a entrada for -2, 4, -4, -6, -11, 4, 18 as saídas deverão ser -2, -4, -6, -11 e 4, 4, 18.

2. Escreva uma função que receba uma lista de números calcule e devolva a média dos elementos da lista. Se a entrada for 8 7 4 19 21 a saída deverá ser 11.8.
3. Escreva uma função que receba uma lista de números contendo uma distribuição qualquer e devolva a distribuição acumulada. Por exemplo, se a entrada for 1, 5, 7, 10, 21 a saída deverá ser 1, 6, 13, 23, 44.
4. Escreva uma função que receba uma lista formada por 3 números (se vierem menos ou mais deve dar um erro e encerrar) e deve imprimir SIM se os números forem lados de um triângulo retângulo e NÃO se não puderem ser. Por exemplo, se a entrada for 3,5,4 a saída deverá ser SIM. Se a entrada for 1,10,100 a saída deverá ser NÃO.
5. Escreva uma função que receba um número inteiro x e devolva True se ele for primo e False senão. Um número é primo quando seus únicos divisores são ele mesmo e a unidade. Se a entrada for 17 a saída deverá ser True.
6. Escreva uma função que receba o valor de x e calcule e devolva o valor de $S = 1/2 + 2/3 + 3/4 + \dots + x/x + 1$. Se a entrada for 5 a saída deverá ser 3.55.
7. Escreva uma função que receba uma lista de números e devolva True se a lista estiver em ordem crescente e False senão. Uma lista X_i está em ordem crescente se $x_{i+1} \geq x_i$. Se a entrada for 1,2,2,2,2,2,3 a saída deverá ser True.
8. Escreva uma função que receba dois números e devolva True se tais números puderem indicar uma hora e minuto e False senão. Por exemplo, se a entrada for 23,59 a saída deverá ser True, idem se for 0,0. Se a entrada for 33,5 a saída deverá ser False.
9. Escreva uma função que receba dois números que descrevem o desempenho de alguém ao cursar a disciplina de MÉTODOS NUMÉRICOS. O primeiro número é a frequência ($0.0 \leq f \leq 100.0$) em porcentagem e o segundo número é a nota final obtida ($0.0 \leq NF \leq 10.0$). Deve imprimir a mensagem APROVADO ou REPROVADO a depender das regras da UFPR para isso. Por exemplo, se $f = 66$ e $NF = 7.1$ a mensagem deve ser REPROVADO.
10. Escreva uma função que receba dois números: o preço do álcool e o preço da gasolina. Sabendo que um carro em particular faz um certo trajeto gastando 10l de álcool e 7l de gasolina (o mesmo trajeto), a função deve recomendar o combustível mais barato, a saber ALCOOL ou GASOLINA. Por exemplo, se $A=2.90$ R\$ e $G=4.10$ R\$, fazendo as contas, a gasolina é mais eficiente e a função deve recomendá-la.



408-75796 - e / nt

Parâmetros opcionais Seja uma função para sublinhar algo. Poderia ser

```
def sublinha():  
    print("_"*30)  
Entretanto, nesta função tanto o caracter sublinha (_ ) quanto o valor 30 são fixos. Para permitir alterar isso, pode-se fazer:  
def sublinha2(n=30, caractere="_"):  
    print(caractere*n)
```

Veja agora como a coisa é chamada

```
>>> def sublinha(n=30, cara="_"):  
    print(cara*n)  
>>> sublinha()  
-----  
>>> sublinha(n=10)
```

```
-----  
>>> sublinha(cara='$')  
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
```

Funções em Python Além das centenas de funções pré-existentes em Python (print, min, max, sum, len, int, float, input...) a grande vantagem do ambiente é que você pode definir as funções que precisar para o seu trabalho. Depois que uma função foi definida ela passa a se comportar como se fosse (e é) uma função primitiva do Python.

Para definir uma nova função, usa-se o comando `def`, veja:

```
def nome(parâmetros):  
    comandos
```

Após a palavra `def` deve vir o nome (único) pelo qual a função será conhecida. Não pode ser uma palavra reservada do Python. Logo depois do nome, é obrigatório um "abre parênteses". A seguir uma quantidade opcional de parâmetros, que são como se fossem nomes de variáveis. Depois desses parâmetros vem o fecha parênteses. Se não houver parâmetro, depois do nome da função vem abre e fecha parênteses. Termina o cabeçalho o caráter "dois pontos" (:). Ele inicia um bloco que limita a definição da função.

Diferentemente do que aconteceu até agora, as linhas que fazem parte da função, não são executadas agora. Elas ficam armazenadas e serão executadas apenas quando a função agora definida, for chamada (executada).

A razão para definir uma função pode ser:

- * Para repetir o mesmo processamento em diversos locais de um programa.
- * Para encapsular um processamento específico.
- * Para comercializar ou apenas trocar algoritmos.
- * Para modularizar um grande processamento, permitindo que o mesmo seja desenvolvido por diversas pessoas.
- * Para poder manusear a complexidade crescente.

A função pode retornar algum resultado para quem a chama. Por exemplo, a função `len(lista)` retorna um valor que corresponde ao comprimento da lista que é passada como parâmetro.

Veja um exemplo de tudo que se disse

```
def maior(a):  
    #1          #1  
    i=len(a)-1  #2  
    mai=-99999  #3  
    while i>= 0: #4  
        if a[i]>mai: #5  
            mai=a[i] #6  
            i=i-1    #7  
    return mai    #8  
    pass         #9
```

```
print(maior([1,5,7,2,3,17,5]))
```

Na linha #1 define-se a função de nome `maior` e se diz que ela deverá ser chamada com um único parâmetro. Quem fez a função sabe que este parâmetro único deve ser uma lista formada por números. Na linha #2 cria-se a variável `i` e atribui-se a ela o valor do tamanho da lista parâmetro menos 1. Como se sabe tal índice acessa o último elemento da lista. Na linha #3 define-se uma outra variável de controle apta a guardar o maior valor da lista até aquele momento. Como ainda nenhum item foi processado ela vale $-\infty$. Na linha #4 define-se que `i` vai variar até ter um valor negativo. Na linha #5 se o elemento de ordem `i` na lista for maior do que a variável `mai`, `mai` será atualizada com este novo valor. Na linha #7 que já está fora da indentação do `if`, mas ainda dentro da indentação do `while` o índice `i` é recuado uma posição e graças ao `while` há um retorno à linha #4. Finalmente, quando o ciclo `while` acabar, vai-se para a linha #8 que simplesmente devolve o valor calculado a quem chamou a função `maior` e o processamento desta função está encerrado. A linha #9 não é executada (de qualquer maneira `pass` não faz nada mesmo), mas lembre-se QUALQUER coisa colocada após um `return` é meramente decorativo, pois nunca vai ser executado. Na linha #10, já fora da função `maior` (note a indentação) a mesma vai ser executada. Veja que preciso colocar um `print` para externalizar o resultado da função. Se não houver `print`, a função vai ser executada, mas nada será visto de fora.

O parâmetro é uma lista ? Se o parâmetro for uma lista de 4 números, ele será um único parâmetro. Estruturalmente é diferente de 4 parâmetros escalares, que no fundo são uma lista de 4 números. Veja

```
def f1(L4):  
    print(L4[0], L4[1], L4[2], L4[3])  
    # ----- OU -----  
def f2(a,b,c,d):  
    print(a, b, c, d)  
f1([1,2,3,4])  
f2(1,2,3,4)
```

Qual estratégia usar ao modularizar um problema e definir quantas e quais funções devem ser construídas é uma arte: a bússola nesta tarefa é a melhor maneira de reaproveitar código e de evitar a reinvenção da roda.

Variáveis locais e globais Como regra geral todas as variáveis definidas dentro de uma função (como `i` e `maior` no exemplo aí acima), são chamadas **locais** à função e isto significa que elas nunca sobrevivem ao tempo de execução da função: não existem antes da função e deixam de existir quando a função acaba. Desta maneira não podem ter seu valor consultado e/ou alterado por alguém externo à função.

Este fato permite que duas funções distintas tenham variáveis com o mesmo nome (como é muito comum um índice chamado `i`) sem que um atrapalhe o outro. Desta maneira a única forma de ceder e receber valores entre funções é através de parâmetros e de retornos.

Já uma variável que precise ser vista e alterada por outros é conhecida como variável **GLOBAL**. Ela deve ser definida dentro de um módulo, mas FORA de uma função. Agora ela pode ser manuseada por todas as funções desse módulo e por todos os módulos que importam o módulo que a definiu.

Para que uma função possa alterar uma variável global, a mesma deve ser definida como global usando-se o comando `GLOBAL` como linha seguinte à definição.

Veja um exemplo:

```
xu = 77  
def fun():  
    xu = 2  
    print(xu)  
print(xu) # 77 é impresso  
fun()     # 2 é impresso  
print(xu) # 77 é impresso  
def alva():  
    global xu  
    xu=2  
alva()  
print(xu) # 2 é impresso
```

As boas práticas de programação sugerem não usar variáveis globais. Esta prática viola o encapsulamento e é fonte frequente de bugs de programação.

Seus módulos podem ser importados Organizando seu trabalho direitinho, você pode modularizar seus programas e importá-los quando necessário, acompanhe

```
#----- beta.py -----  
def alfa(x):  
    return x**3  
#-----  
>>> import beta  
>>> print(beta.alfa(4))  
#-----  
>>> from beta import *  
>>> print(alfa(4))
```

☞ Para você fazer

A seguir uma lista de funções que você deve programar em Python. Imprima as funções e devolva-as impressas junto com esta folha. Não esqueça de identificar com seu nome e o código 3 limite entrega 10/10/24

1. Escreva uma função que receba uma lista de números contendo uma distribuição qualquer e devolva a distribuição acumulada. Por exemplo, se a entrada for 1, 5, 7, 10, 21 a saída deverá ser 1, 6, 13, 23, 44.
2. Uma empresa de eletricidade cobra de seus clientes uma taxa base de R\$ 10 mensais mais seis centavos por quilowatt-hora para

os primeiros 1200 quilowatt-hora e 7 centavos por quilowatt-hora para todo uso acima de 1.200. Escreva uma função que receba o consumo e indique o valor a ser cobrado. Por exemplo, para um consumo de 1000 Kwh, deve cobrar $10+60=70$ R\$.

3. Escreva uma função que receba uma lista de números e retorne a sua média. Por exemplo, se a entrada for 1, 2, 3, 4 a saída deverá ser 2.5.
4. Escreva uma função que receba uma lista de números e devolva o maior deles. Se a entrada for 33, 22, -10, -4 e 33 a saída deve ser 33.
5. Escreva uma função que receba um string contendo uma frase e devolva a quantidade de palavras que formam a frase. Se a entrada for "IVO VIU A UVA" a saída deverá ser 4.
6. Escreva uma função que receba uma lista de números e devolva o menor deles. Se a entrada for 33, 22, -10, -4 e 33 a saída deve ser -10.
7. Escreva uma função que receba um número inteiro x e calcule e devolva a soma dos múltiplos de 13 menores que x . Se a entrada for 100, a saída deverá ser 364 que é igual à soma de 13 26 39 52 65 78 91.
8. Um plano de telefonia celular tem uma taxa de R\$ 35 por mês. O plano inclui 400 minutos gratuitos e taxa de 10 centavos para cada minuto adicional utilizado. Escreva uma função que indique o custo mensal em função do número de minutos utilizados. Por exemplo se a função for chamada com 410, deve responder 36 R\$.
9. Escreva uma função que receba uma lista com 3 valores numéricos: A, B e C e calcule as raízes de $Ax^2 + Bx + c = 0$, usando a fórmula de Baskara. Se a entrada for 1, -5 e 6 a resposta deverá ser 2 e 3.
10. Escreva uma função que receba um valor numérico indicando o lado de um quadrado, medido em metros e devolva o valor do perímetro e da área do quadrado. Se a entrada for 1, a saída deve ser 4 e 1.



408-75808 - e /nt

Parâmetros opcionais Seja uma função para sublinhar algo. Poderia ser

```
def sublinha():  
    print("_"*30)
```

Entretanto, nesta função tanto o caracter sublinha (`_`) quanto o valor 30 são fixos. Para permitir alterar isso, pode-se fazer:

```
def sublinha2(n=30, caractere="_"):  
    print(caractere*n)
```

Veja agora como a coisa é chamada

```
>>> def sublinha(n=30, cara="_"):  
        print(cara*n)  
>>> sublinha()
```

```
-----  
>>> sublinha(n=10)
```

```
-----  
>>> sublinha(cara='$')  
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
```

Funções em Python Além das centenas de funções pré-existentes em Python (print, min, max, sum, len, int, float, input...) a grande vantagem do ambiente é que você pode definir as funções que precisar para o seu trabalho. Depois que uma função foi definida ela passa a se comportar como se fosse (e é) uma função primitiva do Python.

Para definir uma nova função, usa-se o comando `def`, veja:

```
def nome(parâmetros):  
    comandos
```

Após a palavra `def` deve vir o nome (único) pelo qual a função será conhecida. Não pode ser uma palavra reservada do Python. Logo depois do nome, é obrigatório um “abre parênteses”. A seguir uma quantidade opcional de parâmetros, que são como se fossem nomes de variáveis. Depois desses parâmetros vem o fecha parênteses. Se não houver parâmetro, depois do nome da função vem abre e fecha parênteses. Termina o cabeçalho o caráter “dois pontos” (:). Ele inicia um bloco que limita a definição da função.

Diferentemente do que aconteceu até agora, as linhas que fazem parte da função, não são executadas agora. Elas ficam armazenadas e serão executadas apenas quando a função agora definida, for chamada (executada).

A razão para definir uma função pode ser:

- * Para repetir o mesmo processamento em diversos locais de um programa.
- * Para encapsular um processamento específico.
- * Para comercializar ou apenas trocar algoritmos.
- * Para modularizar um grande processamento, permitindo que o mesmo seja desenvolvido por diversas pessoas.
- * Para poder manusear a complexidade crescente.

A função pode retornar algum resultado para quem a chama. Por exemplo, a função `len(lista)` retorna um valor que corresponde ao comprimento da lista que é passada como parâmetro.

Veja um exemplo de tudo que se disse

```
def maior(a):  
    #1          #1  
    i=len(a)-1  #2  
    mai=-99999  #3  
    while i>= 0: #4  
        if a[i]>mai: #5  
            mai=a[i] #6  
        i=i-1 #7  
    return mai #8  
    pass #9
```

```
print(maior([1,5,7,2,3,17,5]))
```

Na linha #1 define-se a função de nome `maior` e se diz que ela deverá ser chamada com um único parâmetro. Quem fez a função sabe que este parâmetro único deve ser uma lista formada por números. Na linha #2 cria-se a variável `i` e atribui-se a ela o valor do tamanho da lista parâmetro menos 1. Como se sabe tal índice acessa o último elemento da lista. Na linha #3 define-se uma outra variável de controle apta a guardar o maior valor da lista até aquele momento. Como ainda nenhum item foi processado ela vale $-\infty$. Na linha #4 define-se que `i` vai variar até ter um valor negativo. Na linha #5 se o elemento de ordem `i` na lista for maior do que a variável `mai`, `mai` será atualizada com este novo valor. Na linha #7 que já está fora da indentação do `if`, mas ainda dentro da indentação do `while` o índice `i` é recuado uma posição e graças ao `while` há um retorno à linha #4. Finalmente, quando o ciclo `while` acabar, vai-se para a linha #8 que simplesmente devolve o valor calculado a quem chamou a função `maior` e o processamento desta função está encerrado. A linha #9 não é executada (de qualquer maneira `pass` não faz nada mesmo), mas lembre-se QUALQUER coisa colocada após um `return` é meramente decorativo, pois nunca vai ser executado. Na linha #10, já fora da função `maior` (note a indentação) a mesma vai ser executada. Veja que preciso colocar um `print` para externalizar o resultado da função. Se não houver `print`, a função vai ser executada, mas nada será visto de fora.

O parâmetro é uma lista ? Se o parâmetro for uma lista de 4 números, ele será um único parâmetro. Estruturalmente é diferente de 4 parâmetros escalares, que no fundo são uma lista de 4 números. Veja

```
def f1(L4):  
    print(L4[0], L4[1], L4[2], L4[3])  
# ----- OU -----  
def f2(a,b,c,d):  
    print(a, b, c, d)  
f1([1,2,3,4])  
f2(1,2,3,4)
```

Qual estratégia usar ao modularizar um problema e definir quantas e quais funções devem ser construídas é uma arte: a bússola nesta tarefa é a melhor maneira de reaproveitar código e de evitar a reinvenção da roda.

Variáveis locais e globais Como regra geral todas as variáveis definidas dentro de uma função (como `i` e `maior` no exemplo aí acima), são chamadas **locais** à função e isto significa que elas nunca sobrevivem ao tempo de execução da função: não existem antes da função e deixam de existir quando a função acaba. Desta maneira não podem ter seu valor consultado e/ou alterado por alguém externo à função.

Este fato permite que duas funções distintas tenham variáveis com o mesmo nome (como é muito comum um índice chamado `i`) sem que um atrapalhe o outro. Desta maneira a única forma de ceder e receber valores entre funções é através de parâmetros e de retornos.

Já uma variável que precise ser vista e alterada por outros é conhecida como variável **GLOBAL**. Ela deve ser definida dentro de um módulo, mas FORA de uma função. Agora ela pode ser manuseada por todas as funções desse módulo e por todos os módulos que importam o módulo que a definiu.

Para que uma função possa alterar uma variável global, a mesma deve ser definida como global usando-se o comando `GLOBAL` como linha seguinte à definição.

Veja um exemplo:

```
xu = 77  
def fun():  
    xu = 2  
    print(xu)  
print(xu) # 77 é impresso  
fun() # 2 é impresso  
print(xu) # 77 é impresso  
def alva():  
    global xu  
    xu=2  
alva()  
print(xu) # 2 é impresso
```

As boas práticas de programação sugerem não usar variáveis globais. Esta prática viola o encapsulamento e é fonte frequente de bugs de programação.

Seus módulos podem ser importados Organizando seu trabalho direitinho, você pode modularizar seus programas e importá-los quando necessário, acompanhe

```
#----- beta.py -----  
def alfa(x):  
    return x**3  
#-----  
>>> import beta  
>>> print(beta.alfa(4))  
#-----  
>>> from beta import *  
>>> print(alfa(4))
```

☞ Para você fazer

A seguir uma lista de funções que você deve programar em Python. Imprima as funções e devolva-as impressas junto com esta folha. Não esqueça de identificar com seu nome e o código 4 limite entrega 10/10/24

1. Escreva uma função que receba uma lista formada por 3 números (se vierem menos ou mais deve dar um erro e encerrar) e deve imprimir SIM se os números forem lados de um triângulo retângulo e NÃO se não puderem ser. Por exemplo, se a entrada for 3,5,4 a saída deverá ser SIM. Se a entrada for 1,10,100 a saída deverá ser NÃO.

2. Escreva uma função que receba um número inteiro e devolva uma lista com os divisores deste número. Se a entrada for 16, a saída deverá ser 1,2,4,8,16.
3. Escreva uma função que receba dois números e devolva `True` se tais números puderem indicar um dia e mês e `False` senão. Por exemplo, se a entrada for 23,11 a saída deverá ser `True`, idem se for 1,1. Se a entrada for 33,5 a saída deverá ser `False`.
4. Escreva uma função que receba uma lista de números e devolva `True` se a lista estiver em ordem crescente e `False` senão. Uma lista X_i está em ordem crescente se $x_{i+1} \geq x_i$. Se a entrada for 1,2,2,2,2,2,3 a saída deverá ser `True`.
5. Escreva uma função que receba um string contendo uma frase e devolva a quantidade de palavras que formam a frase. Se a entrada for "IVO VIU A UVA" a saída deverá ser 4.
6. Escreva uma função que receba 2 valores: o peso e a altura de alguém e devolva o IMC (Índice de Massa Corporal): $IMC = peso \div (altura)^2$. Peso em Kg e altura em m. Se a entrada for 80, 1.80 a saída deverá ser 24.69.
7. Escreva uma função que receba uma lista com 3 valores numéricos: A, B e C e calcule as raízes de $Ax^2 + Bx + c = 0$, usando a fórmula de Baskara. Se a entrada for 1, -5 e 6 a resposta deverá ser 2 e 3.
8. Escreva uma função que receba um número inteiro `x` e devolva `True` se ele for primo e `False` senão. Um número é primo quando seus únicos divisores são ele mesmo e a unidade. Se a entrada for 17 a saída deverá ser `True`.
9. Escreva uma função que receba dois números: o preço do álcool e o preço da gasolina. Sabendo que um carro em particular faz um certo trajeto gastando 10l de álcool e 7l de gasolina (o mesmo trajeto), a função deve recomendar o combustível mais barato, a saber ALCOOL ou GASOLINA. Por exemplo, se A=2.90 R\$ e G=4.10 R\$, fazendo as contas, a gasolina é mais eficiente e a função deve recomendá-la.
10. Escreva uma função que receba um parâmetro numérico e retorne a palavra PAR ou a palavra IMPAR se o parâmetro for par ou ímpar. Por exemplo, se a entrada for 3, a saída deverá ser IMPAR.



408-75815 - e /nt

Parâmetros opcionais Seja uma função para sublinhar algo. Poderia ser

```
def sublinha():  
    print("_"*30)
```

Entretanto, nesta função tanto o caracter sublinha (`_`) quanto o valor 30 são fixos. Para permitir alterar isso, pode-se fazer:

```
def sublinha2(n=30, caractere="_"):  
    print(caractere*n)
```

Veja agora como a coisa é chamada

```
>>> def sublinha(n=30, cara="_"):  
    print(cara*n)  
>>> sublinha()
```

```
-----  
>>> sublinha(n=10)
```

```
-----  
>>> sublinha(cara='$')  
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
```

Funções em Python Além das centenas de funções pré-existentes em Python (print, min, max, sum, len, int, float, input...) a grande vantagem do ambiente é que você pode definir as funções que precisar para o seu trabalho. Depois que uma função foi definida ela passa a se comportar como se fosse (e é) uma função primitiva do Python.

Para definir uma nova função, usa-se o comando `def`, veja:

```
def nome(parâmetros):  
    comandos
```

Após a palavra `def` deve vir o nome (único) pelo qual a função será conhecida. Não pode ser uma palavra reservada do Python. Logo depois do nome, é obrigatório um "abre parênteses". A seguir uma quantidade opcional de parâmetros, que são como se fossem nomes de variáveis. Depois desses parâmetros vem o fecha parênteses. Se não houver parâmetro, depois do nome da função vem abre e fecha parênteses. Termina o cabeçalho o caráter "dois pontos" (:). Ele inicia um bloco que limita a definição da função.

Diferentemente do que aconteceu até agora, as linhas que fazem parte da função, não são executadas agora. Elas ficam armazenadas e serão executadas apenas quando a função agora definida, for chamada (executada).

A razão para definir uma função pode ser:

- * Para repetir o mesmo processamento em diversos locais de um programa.
- * Para encapsular um processamento específico.
- * Para comerciar ou apenas trocar algoritmos.
- * Para modularizar um grande processamento, permitindo que o mesmo seja desenvolvido por diversas pessoas.
- * Para poder manusear a complexidade crescente.

A função pode retornar algum resultado para quem a chama. Por exemplo, a função `len(lista)` retorna um valor que corresponde ao comprimento da lista que é passada como parâmetro.

Veja um exemplo de tudo que se disse

```
def maior(a):  
    #1  
    i=len(a)-1 #2  
    mai=-99999 #3  
    while i>= 0:  
        if a[i]>mai: #5  
            mai=a[i] #6  
            i=i-1 #7  
    return mai #8  
    pass #9
```

```
print(maior([1,5,7,2,3,17,5])) #10
```

Na linha #1 define-se a função de nome `maior` e se diz que ela deverá ser chamada com um único parâmetro. Quem fez a função sabe que este parâmetro único deve ser uma lista formada por números. Na linha #2 cria-se a variável `i` e atribui-se a ela o valor do tamanho da lista parâmetro menos 1. Como se sabe tal índice acessa o último elemento da lista. Na linha #3 define-se uma outra variável de controle apta a guardar o maior valor da lista até aquele momento. Como ainda nenhum item foi processado ela vale $-\infty$. Na linha #4 define-se que `i` vai variar até ter um valor negativo. Na linha #5 se o elemento de ordem `i` na lista for maior do que a variável `mai`, `mai` será atualizada com este novo valor. Na linha #7 que já está fora da indentação do `if`, mas ainda dentro da indentação do `while` o índice `i` é recuado uma posição e graças ao `while` há um retorno à linha #4. Finalmente, quando o ciclo `while` acabar, vai-se para a linha #8 que simplesmente devolve o valor calculado a quem chamou a função `maior` e o processamento desta função está encerrado. A linha #9 não é executada (de qualquer maneira `pass` não faz nada mesmo), mas lembre-se QUALQUER coisa colocada após um `return` é meramente decorativo, pois nunca vai ser executado. Na linha #10, já fora da função `maior` (note a indentação) a mesma vai ser executada. Veja que preciso colocar um `print` para externalizar o resultado da função. Se não houver `print`, a função vai ser executada, mas nada será visto de fora.

O parâmetro é uma lista ? Se o parâmetro for uma lista de 4 números, ele será um único parâmetro. Estruturalmente é diferente de 4 parâmetros escalares, que no fundo são uma lista de 4 números. Veja

```
def f1(L4):  
    print(L4[0], L4[1], L4[2], L4[3])  
# ---- OU ----  
def f2(a,b,c,d):  
    print(a, b, c, d)  
f1([1,2,3,4])  
f2(1,2,3,4)
```

Qual estratégia usar ao modularizar um problema e definir quantas e quais funções devem ser construídas é uma arte: a bússola nesta tarefa é a melhor maneira de reaproveitar código e de evitar a reinvenção da roda.

Variáveis locais e globais Como regra geral todas as variáveis definidas dentro de uma função (como `i` e `maior` no exemplo aí acima), são chamadas **locais** à função e isto significa que elas nunca sobrevivem ao tempo de execução da função: não existem antes da função e deixam de existir quando a função acaba. Desta maneira não podem ter seu valor consultado e/ou alterado por alguém externo à função.

Este fato permite que duas funções distintas tenham variáveis com o mesmo nome (como é muito comum um índice chamado `i`) sem que um atrapalhe o outro. Desta maneira a única forma de ceder e receber valores entre funções é através de parâmetros e de retornos.

Já uma variável que precise ser vista e alterada por outros é conhecida como variável GLOBAL. Ela deve ser definida dentro de um módulo, mas FORA de uma função. Agora ela pode ser manuseada por todas as funções desse módulo e por todos os módulos que importam o módulo que a definiu.

Para que uma função possa alterar uma variável global, a mesma deve ser definida como global usando-se o comando GLOBAL como linha seguinte à definição.

Veja um exemplo:

```
xu = 77  
def fun():  
    xu = 2  
    print(xu)  
print(xu) # 77 é impresso  
fun() # 2 é impresso  
print(xu) # 77 é impresso  
def alva():  
    global xu  
    xu=2  
alva()  
print(xu) # 2 é impresso
```

As boas práticas de programação sugerem não usar variáveis globais. Esta prática viola o encapsulamento e é fonte frequente de bugs de programação.

Seus módulos podem ser importados Organizando seu trabalho direitinho, você pode modularizar seus programas e importá-los quando necessário, acompanhe

```
#----- beta.py -----  
def alfa(x):  
    return x**3  
#-----  
>>> import beta  
>>> print(beta.alfa(4))  
#-----  
>>> from beta import *  
>>> print(alfa(4))
```

👉 Para você fazer

A seguir uma lista de funções que você deve programar em Python. Imprima as funções e devolva-as impressas junto com esta folha. Não esqueça de identificar com seu nome e o código 5 limite entrega 10/10/24

1. Escreva uma função que receba uma lista de números e devolva a quantidade de números que são menores do que seus vizinhos. Os extremos da lista não devem ser testados pois lhes falta algum vizinho. Se a entrada for 1,4,2,8,22,50,100 a saída deverá ser 1.

2. Uma empresa de eletricidade cobra de seus clientes uma taxa base de R\$ 10 mensais mais seis centavos por quilowatt-hora para os primeiros 1200 quilowatt-hora e 7 centavos por quilowatt-hora para todo uso acima de 1.200. Escreva uma função que receba o consumo e indique o valor a ser cobrado. Por exemplo, para um consumo de 1000 Kwh, deve cobrar $10+60=70$ R\$.
3. Escreva uma função que receba dois números e devolva `True` se tais números puderem indicar uma hora e minuto e `False` senão. Por exemplo, se a entrada for 23,59 a saída deverá ser `True`, idem se for 0,0. Se a entrada for 33,5 a saída deverá ser `False`.
4. Escreva uma função que receba a altura e o peso de uma pessoa. Deve calcular o IMC - índice de massa corporal ($IMV = \text{peso}/\text{altura}^2$) medidos em kg e m. A seguir deve emitir mensagem a depender do IMC. $IMC < 25$ =adequado; $IMC < 30$ =sobrepeso; $IMC < 35$ =obesidade e $IMC \geq 35$ =obesidade severa. Por exemplo, se o sujeito pesa 110Kg e tem 1.70m de altura seu IMC é de 38.06. Logo o diagnóstico é OBESIDADE SEVERA.
5. Escreva uma função que receba uma lista de números e devolva o menor deles. Se a entrada for 33, 22, -10, -4 e 33 a saída deve ser -10.
6. Escreva uma função que receba dois números e devolva `True` se tais números puderem indicar um dia e mês e `False` senão. Por exemplo, se a entrada for 23,11 a saída deverá ser `True`, idem se for 1,1. Se a entrada for 33,5 a saída deverá ser `False`.
7. Escreva uma função que receba um número inteiro `x` e calcule e devolva a soma dos múltiplos de 13 menores que `x`. Se a entrada for 100, a saída deverá ser 364 que é igual à soma de 13 26 39 52 65 78 91.
8. Escreva uma função que receba uma lista de números e devolva a diferença entre o maior e o menor deles. Se a entrada for 33, 22, -10, -4 e 33 a saída deve ser 43.
9. Escreva uma função que receba uma lista de números e devolva o maior deles. Se a entrada for 33, 22, -10, -4 e 33 a saída deve ser 33.
10. Escreva uma função que receba uma lista de números calcule e devolva a média dos elementos da lista. Se a entrada for 8 7 4 19 21 a saída deverá ser 11.8.



408-75822 - e /nt

Parâmetros opcionais Seja uma função para sublinhar algo. Poderia ser

```
def sublinha():  
    print("_"*30)
```

Entretanto, nesta função tanto o caracter sublinha (`_`) quanto o valor 30 são fixos. Para permitir alterar isso, pode-se fazer:

```
def sublinha2(n=30, caractere="_"):  
    print(caractere*n)
```

Veja agora como a coisa é chamada

```
>>> def sublinha(n=30, cara="_"):  
    print(cara*n)  
>>> sublinha()
```

```
>>> sublinha(n=10)
```

```
>>> sublinha(cara='$')  
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
```

Funções em Python Além das centenas de funções pré-existentes em Python (print, min, max, sum, len, int, float, input...) a grande vantagem do ambiente é que você pode definir as funções que precisar para o seu trabalho. Depois que uma função foi definida ela passa a se comportar como se fosse (e é) uma função primitiva do Python.

Para definir uma nova função, usa-se o comando `def`, veja:

```
def nome(parâmetros):  
    comandos
```

Após a palavra `def` deve vir o nome (único) pelo qual a função será conhecida. Não pode ser uma palavra reservada do Python. Logo depois do nome, é obrigatório um "abre parênteses". A seguir uma quantidade opcional de parâmetros, que são como se fossem nomes de variáveis. Depois desses parâmetros vem o fecha parênteses. Se não houver parâmetro, depois do nome da função vem abre e fecha parênteses. Termina o cabeçalho o caráter "dois pontos" (:). Ele inicia um bloco que limita a definição da função.

Diferentemente do que aconteceu até agora, as linhas que fazem parte da função, não são executadas agora. Elas ficam armazenadas e serão executadas apenas quando a função agora definida, for chamada (executada).

A razão para definir uma função pode ser:

- * Para repetir o mesmo processamento em diversos locais de um programa.
- * Para encapsular um processamento específico.
- * Para comerciar ou apenas trocar algoritmos.
- * Para modularizar um grande processamento, permitindo que o mesmo seja desenvolvido por diversas pessoas.
- * Para poder manusear a complexidade crescente.

A função pode retornar algum resultado para quem a chama. Por exemplo, a função `len(lista)` retorna um valor que corresponde ao comprimento da lista que é passada como parâmetro.

Veja um exemplo de tudo que se disse

```
def maior(a):  
    #1  
    i=len(a)-1 #2  
    mai=-99999 #3  
    while i>= 0:  
        #4  
        if a[i]>mai:  
            #5  
            mai=a[i] #6  
        i=i-1 #7  
    return mai #8  
    pass #9
```

```
print(maior([1,5,7,2,3,17,5])) #10
```

Na linha #1 define-se a função de nome `maior` e se diz que ela deverá ser chamada com um único parâmetro. Quem fez a função sabe que este parâmetro único deve ser uma lista formada por números. Na linha #2 cria-se a variável `i` e atribui-se a ela o valor do tamanho da lista parâmetro menos 1. Como se sabe tal índice acessa o último elemento da lista. Na linha #3 define-se uma outra variável de controle apta a guardar o maior valor da lista até aquele momento. Como ainda nenhum item foi processado ela vale $-\infty$. Na linha #4 define-se que `i` vai variar até ter um valor negativo. Na linha #5 se o elemento de ordem `i` na lista for maior do que a variável `mai`, `mai` será atualizada com este novo valor. Na linha #7 que já está fora da indentação do `if`, mas ainda dentro da indentação do `while` o índice `i` é recuado uma posição e graças ao `while` há um retorno à linha #4. Finalmente, quando o ciclo `while` acabar, vai-se para a linha #8 que simplesmente devolve o valor calculado a quem chamou a função `maior` e o processamento desta função está encerrado. A linha #9 não é executada (de qualquer maneira `pass` não faz nada mesmo), mas lembre-se QUALQUER coisa colocada após um `return` é meramente decorativo, pois nunca vai ser executado. Na linha #10, já fora da função `maior` (note a indentação) a mesma vai ser executada. Veja que preciso colocar um `print` para externalizar o resultado da função. Se não houver `print`, a função vai ser executada, mas nada será visto de fora.

O parâmetro é uma lista ? Se o parâmetro for uma lista de 4 números, ele será um único parâmetro. Estruturalmente é diferente de 4 parâmetros escalares, que no fundo são uma lista de 4 números. Veja

```
def f1(L4):  
    print(L4[0], L4[1], L4[2], L4[3])  
    # ---- OU ----  
def f2(a,b,c,d):  
    print(a, b, c, d)  
f1([1,2,3,4])  
f2(1,2,3,4)
```

Qual estratégia usar ao modularizar um problema e definir quantas e quais funções devem ser construídas é uma arte: a bússola nesta tarefa é a melhor maneira de reaproveitar código e de evitar a reinvenção da roda.

Variáveis locais e globais Como regra geral todas as variáveis definidas dentro de uma função (como `i` e `maior` no exemplo aí acima), são chamadas **locais** à função e isto significa que elas nunca sobrevivem ao tempo de execução da função: não existem antes da função e deixam de existir quando a função acaba. Desta maneira não podem ter seu valor consultado e/ou alterado por alguém externo à função.

Este fato permite que duas funções distintas tenham variáveis com o mesmo nome (como é muito comum um índice chamado `i`) sem que um atrapalhe o outro. Desta maneira a única forma de ceder e receber valores entre funções é através de parâmetros e de retornos.

Já uma variável que precise ser vista e alterada por outros é conhecida como variável **GLOBAL**. Ela deve ser definida dentro de um módulo, mas **FORA** de uma função. Agora ela pode ser manuseada por todas as funções desse módulo e por todos os módulos que importam o módulo que a definiu.

Para que uma função possa alterar uma variável global, a mesma deve ser definida como global usando-se o comando **GLOBAL** como linha seguinte à definição.

Veja um exemplo:

```
xu = 77  
def fun():  
    xu = 2  
    print(xu)  
print(xu) # 77 é impresso  
fun() # 2 é impresso  
print(xu) # 77 é impresso  
def alva():  
    global xu  
    xu=2  
alva()  
print(xu) # 2 é impresso
```

As boas práticas de programação sugerem não usar variáveis globais. Esta prática viola o encapsulamento e é fonte frequente de bugs de programação.

Seus módulos podem ser importados Organizando seu trabalho direitinho, você pode modularizar seus programas e importá-los quando necessário, acompanhe

```
#----- beta.py -----  
def alfa(x):  
    return x**3  
#-----  
>>> import beta  
>>> print(beta.alfa(4))  
#-----  
>>> from beta import *  
>>> print(alfa(4))
```

📖 Para você fazer

A seguir uma lista de funções que você deve programar em Python. Imprima as funções e devolva-as impressas junto com esta folha. Não esqueça de identificar com seu nome e o código 6 limite entrega 10/10/24

1. Escreva uma função que receba uma lista de números e devolva `True` se a lista estiver em ordem crescente e `False` senão. Uma lista X_i está em ordem crescente se $x_{i+1} \geq x_i$. Se a entrada for 1,2,2,2,2,3 a saída deverá ser `True`.

2. Um plano de telefonia celular tem uma taxa de R\$ 35 por mês. O plano inclui 400 minutos gratuitos e taxa de 10 centavos para cada minuto adicional utilizado. Escreva uma função que indique o custo mensal em função do número de minutos utilizados. Por exemplo se a função for chamada com 410, deve responder 36 R\$.
3. Escreva uma função que receba um valor numérico indicando o lado de um quadrado, medido em metros e devolva o valor do perímetro e da área do quadrado. Se a entrada for 1, a saída deve ser 4 e 1.
4. Escreva uma função que receba a altura e o peso de uma pessoa. Deve calcular o IMC - índice de massa corporal ($IMV = \text{peso}/\text{altura}^2$) medidos em kg e m. A seguir deve emitir mensagem a depender do IMC. $IMC < 25$ =adequado; $IMC < 30$ =sobrepeso; $IMC < 35$ =obesidade e $IMC \geq 35$ =obesidade severa. Por exemplo, se o sujeito pesa 110Kg e tem 1.70m de altura seu IMC é de 38.06. Logo o diagnóstico é **OBESIDADE SEVERA**.
5. Escreva uma função que receba um número inteiro `x` e devolva `True` se ele for primo e `False` senão. Um número é primo quando seus únicos divisores são ele mesmo e a unidade. Se a entrada for 17 a saída deverá ser `True`.
6. Escreva uma função que receba o valor de `x` e calcule e devolva o valor de $S = 1/4 + 2/9 + 3/16 + \dots + x/(x+1)^2$. Se a entrada for 5 a saída deverá ser 0.958611111.
7. Escreva uma função que receba o valor de `x` e calcule e devolva o valor de $S = 1/2 + 2/3 + 3/4 + \dots + x/x + 1$. Se a entrada for 5 a saída deverá ser 3.55.
8. Escreva uma função que receba um número inteiro e devolva a soma de todos os divisores deste número. Se a entrada for 15 a saída deverá ser 24.
9. Escreva uma função que receba um string contendo uma frase e devolva a quantidade de vogais que foram usadas na frase. Se a entrada for "Ivo viu a uva" a saída deverá ser 7.
10. Escreva uma função que receba uma lista de números e devolva a quantidade de números que são menores do que seus vizinhos. Os limites da lista não devem ser testados pois lhes falta algum vizinho. Se a entrada for 1,4,2,8,22,50,100 a saída deverá ser 1.



408-75839 - e /nt

Parâmetros opcionais Seja uma função para sublinhar algo. Poderia ser

```
def sublinha():  
    print("_"*30)  
Entretanto, nesta função tanto o caracter sublinha (_) quanto o valor 30 são fixos. Para permitir alterar isso, pode-se fazer:  
def sublinha2(n=30, caractere="_"):  
    print(caractere*n)  
Veja agora como a coisa é chamada  
>>> def sublinha(n=30, cara="_"):  
        print(cara*n)  
>>> sublinha()  
-----  
>>> sublinha(n=10)  
-----  
>>> sublinha(cara='$')  
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
```

Funções em Python Além das centenas de funções pré-existentes em Python (print, min, max, sum, len, int, float, input...) a grande vantagem do ambiente é que você pode definir as funções que precisar para o seu trabalho. Depois que uma função foi definida ela passa a se comportar como se fosse (e é) uma função primitiva do Python.

Para definir uma nova função, usa-se o comando def, veja:

```
def nome(parâmetros):  
    comandos
```

Após a palavra def deve vir o nome (único) pelo qual a função será conhecida. Não pode ser uma palavra reservada do Python. Logo depois do nome, é obrigatório um "abre parênteses". A seguir uma quantidade opcional de parâmetros, que são como se fossem nomes de variáveis. Depois desses parâmetros vem o fecha parênteses. Se não houver parâmetro, depois do nome da função vem abre e fecha parênteses. Termina o cabeçalho o caráter "dois pontos" (:). Ele inicia um bloco que limita a definição da função.

Diferentemente do que aconteceu até agora, as linhas que fazem parte da função, não são executadas agora. Elas ficam armazenadas e serão executadas apenas quando a função agora definida, for chamada (executada).

A razão para definir uma função pode ser:

- * Para repetir o mesmo processamento em diversos locais de um programa.
- * Para encapsular um processamento específico.
- * Para comercializar ou apenas trocar algoritmos.
- * Para modularizar um grande processamento, permitindo que o mesmo seja desenvolvido por diversas pessoas.
- * Para poder manusear a complexidade crescente.

A função pode retornar algum resultado para quem a chama. Por exemplo, a função len(lista) retorna um valor que corresponde ao comprimento da lista que é passada como parâmetro.

Veja um exemplo de tudo que se disse

```
def maior(a):  
    i=len(a)-1 #1  
    mai=-99999 #2  
    while i>= 0: #3  
        if a[i]>mai: #4  
            mai=a[i] #5  
            i=i-1 #6  
    return mai #7  
    pass #8  
print(maior([1,5,7,2,3,17,5])) #9
```

Na linha #1 define-se a função de nome maior e se diz que ela deverá ser chamada com um único parâmetro. Quem fez a função sabe que este parâmetro único deve ser uma lista formada por números. Na linha #2 cria-se a variável i e atribui-se a ela o valor do tamanho da lista parâmetro menos 1. Como se sabe tal índice acessa o último elemento da lista. Na linha #3 define-se uma outra variável de controle apta a guardar o maior valor da lista até aquele momento. Como ainda nenhum item foi processado ela vale -∞. Na linha #4 define-se que i vai variar até ter um valor negativo. Na linha #5 se o elemento de ordem i na lista for maior do que a variável mai, mai será atualizada com este novo valor. Na linha #7 que já está fora da indentação do if, mas ainda dentro da indentação do while o índice i é recuado uma posição e graças ao while há um retorno à linha #4. Finalmente, quando o ciclo while acabar, vai-se para a linha #8 que simplesmente devolve o valor calculado a quem chamou a função maior e o processamento desta função está encerrado. A linha #9 não é executada (de qualquer maneira pass não faz nada mesmo), mas lembre-se QUALQUER coisa colocada após um return é meramente decorativo, pois nunca vai ser executado. Na linha #10, já fora da função maior (note a indentação) a mesma vai ser executada. Veja que preciso colocar um print para externalizar o resultado da função. Se não houver print, a função vai ser executada, mas nada será visto de fora.

O parâmetro é uma lista ? Se o parâmetro for uma lista de 4 números, ele será um único parâmetro. Estruturalmente é diferente de 4 parâmetros escalares, que no fundo são uma lista de 4 números. Veja

```
def f1(L4):  
    print(L4[0], L4[1], L4[2], L4[3])  
# ----- OU -----  
def f2(a,b,c,d):  
    print(a, b, c, d)  
f1([1,2,3,4])  
f2(1,2,3,4)
```

Qual estratégia usar ao modularizar um problema e definir quantas e quais funções devem ser construídas é uma arte: a bússola nesta tarefa é a melhor maneira de reaproveitar código e de evitar a reinvenção da roda.

Variáveis locais e globais Como regra geral todas as variáveis definidas dentro de uma função (como i e maior no exemplo aí acima), são chamadas locais à função e isto significa que elas nunca sobrevivem ao tempo de execução da função: não existem antes da função e deixam de existir quando a função acaba. Desta maneira não podem ter seu valor consultado e/ou alterado por alguém externo à função.

Este fato permite que duas funções distintas tenham variáveis com o mesmo nome (como é muito comum um índice chamado i) sem que um atrapalhe o outro. Desta maneira a única forma de ceder e receber valores entre funções é através de parâmetros e de retornos.

Já uma variável que precise ser vista e alterada por outros é conhecida como variável GLOBAL. Ela deve ser definida dentro de um módulo, mas FORA de uma função. Agora ela pode ser manuseada por todas as funções desse módulo e por todos os módulos que importam o módulo que a definiu.

Para que uma função possa alterar uma variável global, a mesma deve ser definida como global usando-se o comando GLOBAL como linha seguinte à definição.

Veja um exemplo:

```
xu = 77  
def fun():  
    xu = 2  
    print(xu)  
print(xu) # 77 é impresso  
fun() # 2 é impresso  
print(xu) # 77 é impresso  
def alva():  
    global xu  
    xu=2  
alva()  
print(xu) # 2 é impresso
```

As boas práticas de programação sugerem não usar variáveis globais. Esta prática viola o encapsulamento e é fonte frequente de bugs de programação.

Seus módulos podem ser importados Organizando seu trabalho direitinho, você pode modularizar seus programas e importá-los quando necessário, acompanhe

```
#----- beta.py -----  
def alfa(x):  
    return x**3  
#-----  
>>> import beta  
>>> print(beta.alfa(4))  
#-----  
>>> from beta import *  
>>> print(alfa(4))
```

☞ Para você fazer

A seguir uma lista de funções que você deve programar em Python. Imprima as funções e devolva-as impressas junto com esta folha. Não esqueça de identificar com seu nome e o código 7 limite entrega 10/10/24

1. Escreva uma função que receba um string contendo uma frase e devolva a quantidade de palavras masculinas que formam a frase. Por hipótese, as palavras masculinas são as que terminam em "O" ou "o". Se a entrada for "O RATO ROEU A ROUPA DO REI" a saída deverá ser 3.

2. Escreva uma função que receba uma lista de números contendo uma distribuição qualquer e devolva a distribuição acumulada. Por exemplo, se a entrada for 1, 5, 7, 10, 21 a saída deverá ser 1, 6, 13, 23, 44.
3. Escreva uma função que receba uma lista formada por 3 números (se vierem menos ou mais deve dar um erro e encerrar) e deve imprimir SIM se os números forem lados de um triângulo retângulo e NÃO se não puderem ser. Por exemplo, se a entrada for 3,5,4 a saída deverá ser SIM. Se a entrada for 1,10,100 a saída deverá ser NÃO.
4. Escreva uma função que receba uma lista de números positivos e negativos misturados e devolva duas listas: a primeira contendo apenas os positivos e a segunda apenas os negativos, na mesma ordem em que eles apareceram na entrada. Se a entrada for -2, 4, -4, -6, -11, 4, 18 as saídas deverão ser -2, -4, -6, -11 e 4, 4, 18.
5. Escreva uma função que receba uma lista de números calcule e devolva a média dos elementos da lista. Se a entrada for 8 7 4 19 21 a saída deverá ser 11.8.
6. Escreva uma função que receba um número inteiro x e devolva True se ele for primo e False senão. Um número é primo quando seus únicos divisores são ele mesmo e a unidade. Se a entrada for 17 a saída deverá ser True.
7. Escreva uma função que receba uma lista de números e devolva a diferença entre o maior e o menor deles. Se a entrada for 33, 22, -10, -4 e 33 a saída deve ser 43.
8. Uma empresa de eletricidade cobra de seus clientes uma taxa base de R\$ 10 mensais mais seis centavos por quilowatt-hora para os primeiros 1200 quilowatt-hora e 7 centavos por quilowatt-hora para todo uso acima de 1.200. Escreva uma função que receba o consumo e indique o valor a ser cobrado. Por exemplo, para um consumo de 1000 Kwh, deve cobrar 10+60=70 R\$.
9. Escreva uma função que receba dois números: o preço do álcool e o preço da gasolina. Sabendo que um carro em particular faz um certo trajeto gastando 10l de álcool e 7l de gasolina (o mesmo trajeto), a função deve recomendar o combustível mais barato, a saber ALCOOL ou GASOLINA. Por exemplo, se A=2.90 R\$ e G=4.10 R\$, fazendo as contas, a gasolina é mais eficiente e a função deve recomendá-la.
10. Escreva uma função que receba um número inteiro e devolva a soma de todos os divisores deste número. Se a entrada for 15 a saída deverá ser 24.



408-75846 - e /nt

Parâmetros opcionais Seja uma função para sublinhar algo. Poderia ser

```
def sublinha():  
    print("_"*30)  
Entretanto, nesta função tanto o caracter sublinha (_ ) quanto o valor 30 são fixos. Para permitir alterar isso, pode-se fazer:  
def sublinha2(n=30, caractere="_"):  
    print(caractere*n)  
Veja agora como a coisa é chamada  
>>> def sublinha(n=30, cara="_"):  
    print(cara*n)  
>>> sublinha()  
-----  
>>> sublinha(n=10)
```

```
-----  
>>> sublinha(cara='$!')  
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
```

Funções em Python Além das centenas de funções pré-existentes em Python (print, min, max, sum, len, int, float, input...) a grande vantagem do ambiente é que você pode definir as funções que precisar para o seu trabalho. Depois que uma função foi definida ela passa a se comportar como se fosse (e é) uma função primitiva do Python.

Para definir uma nova função, usa-se o comando `def`, veja:

```
def nome(parâmetros):  
    comandos
```

Após a palavra `def` deve vir o nome (único) pelo qual a função será conhecida. Não pode ser uma palavra reservada do Python. Logo depois do nome, é obrigatório um "abre parênteses". A seguir uma quantidade opcional de parâmetros, que são como se fossem nomes de variáveis. Depois desses parâmetros vem o fecha parênteses. Se não houver parâmetro, depois do nome da função vem abre e fecha parênteses. Termina o cabeçalho o caráter "dois pontos" (:). Ele inicia um bloco que limita a definição da função.

Diferentemente do que aconteceu até agora, as linhas que fazem parte da função, não são executadas agora. Elas ficam armazenadas e serão executadas apenas quando a função agora definida, for chamada (executada).

A razão para definir uma função pode ser:

- * Para repetir o mesmo processamento em diversos locais de um programa.
- * Para encapsular um processamento específico.
- * Para comerciar ou apenas trocar algoritmos.
- * Para modularizar um grande processamento, permitindo que o mesmo seja desenvolvido por diversas pessoas.
- * Para poder manusear a complexidade crescente.

A função pode retornar algum resultado para quem a chama. Por exemplo, a função `len(lista)` retorna um valor que corresponde ao comprimento da lista que é passada como parâmetro.

Veja um exemplo de tudo que se disse

```
def maior(a):  
    i=len(a)-1 #1  
    mai=-99999 #2  
    while i>= 0: #3  
        if a[i]>mai: #4  
            mai=a[i] #5  
            i=i-1 #6  
    return mai #7  
    pass #8  
print(maior([1,5,7,2,3,17,5])) #9
```

Na linha #1 define-se a função de nome `maior` e se diz que ela deverá ser chamada com um único parâmetro. Quem fez a função sabe que este parâmetro único deve ser uma lista formada por números. Na linha #2 cria-se a variável `i` e atribui-se a ela o valor do tamanho da lista parâmetro menos 1. Como se sabe tal índice acessa o último elemento da lista. Na linha #3 define-se uma outra variável de controle apta a guardar o maior valor da lista até aquele momento. Como ainda nenhum item foi processado ela vale $-\infty$. Na linha #4 define-se que `i` vai variar até ter um valor negativo. Na linha #5 se o elemento de ordem `i` na lista for maior do que a variável `mai`, `mai` será atualizada com este novo valor. Na linha #7 que já está fora da indentação do `if`, mas ainda dentro da indentação do `while` o índice `i` é recuado uma posição e graças ao `while` há um retorno à linha #4. Finalmente, quando o ciclo `while` acabar, vai-se para a linha #8 que simplesmente devolve o valor calculado a quem chamou a função `maior` e o processamento desta função está encerrado. A linha #9 não é executada (de qualquer maneira `pass` não faz nada mesmo), mas lembre-se QUALQUER coisa colocada após um `return` é meramente decorativo, pois nunca vai ser executado. Na linha #10, já fora da função `maior` (note a indentação) a mesma vai ser executada. Veja que preciso colocar um `print` para externalizar o resultado da função. Se não houver `print`, a função vai ser executada, mas nada será visto de fora.

O parâmetro é uma lista ? Se o parâmetro for uma lista de 4 números, ele será um único parâmetro. Estruturalmente é diferente de 4 parâmetros escalares, que no fundo são uma lista de 4 números. Veja

```
def f1(L4):  
    print(L4[0], L4[1], L4[2], L4[3])  
# ----- OU -----  
def f2(a,b,c,d):  
    print(a, b, c, d)  
f1([1,2,3,4])  
f2(1,2,3,4)
```

Qual estratégia usar ao modularizar um problema e definir quantas e quais funções devem ser construídas é uma arte: a bússola nesta tarefa é a melhor maneira de reaproveitar código e de evitar a reinvenção da roada.

Variáveis locais e globais Como regra geral todas as variáveis definidas dentro de uma função (como `i` e `maior` no exemplo aí acima), são chamadas **locais** à função e isto significa que elas nunca sobrevivem ao tempo de execução da função: não existem antes da função e deixam de existir quando a função acaba. Desta maneira não podem ter seu valor consultado e/ou alterado por alguém externo à função.

Este fato permite que duas funções distintas tenham variáveis com o mesmo nome (como é muito comum um índice chamado `i`) sem que um atrapalhe o outro. Desta maneira a única forma de ceder e receber valores entre funções é através de parâmetros e de retornos.

Já uma variável que precise ser vista e alterada por outros é conhecida como variável **GLOBAL**. Ela deve ser definida dentro de um módulo, mas **FORA** de uma função. Agora ela pode ser manuseada por todas as funções desse módulo e por todos os módulos que importam o módulo que a definiu.

Para que uma função possa alterar uma variável global, a mesma deve ser definida como global usando-se o comando `GLOBAL` como linha seguinte à definição.

Veja um exemplo:

```
xu = 77  
def fun():  
    xu = 2  
    print(xu)  
print(xu) # 77 é impresso  
fun() # 2 é impresso  
print(xu) # 77 é impresso  
def alva():  
    global xu  
    xu=2  
alva()  
print(xu) # 2 é impresso
```

As boas práticas de programação sugerem não usar variáveis globais. Esta prática viola o encapsulamento e é fonte frequente de bugs de programação.

Seus módulos podem ser importados Organizando seu trabalho direitinho, você pode modularizar seus programas e importá-los quando necessário, acompanhe

```
#----- beta.py -----  
def alfa(x):  
    return x**3  
#-----  
>>> import beta  
>>> print(beta.alfa(4))  
#-----  
>>> from beta import *  
>>> print(alfa(4))
```

☞ Para você fazer

A seguir uma lista de funções que você deve programar em Python. Imprima as funções e devolva-as impressas junto com esta folha. Não esqueça de identificar com seu nome e o código 8 limite entrega 10/10/24

1. Escreva uma função que receba uma lista formada por 3 números (se vierem menos ou mais deve dar um erro e encerrar) e deve imprimir SIM se os números forem lados de um triângulo retângulo e NÃO se não puderem ser. Por exemplo, se a entrada for 3,5,4 a saída deverá ser SIM. Se a entrada for 1,10,100 a saída deverá ser NÃO.

2. Escreva uma função que receba a altura e o peso de uma pessoa. Deve calcular o IMC - índice de massa corporal ($IMV = \text{peso}/\text{altura}^2$) medidos em kg e m. A seguir deve emitir mensagem a depender do IMC. $IMC < 25$ =adequado; $IMC < 30$ =sobrepeso; $IMC < 35$ =obesidade e $IMC \geq 35$ =obesidade severa. Por exemplo, se o sujeito pesa 110Kg e tem 1.70m de altura seu IMC é de 38.06. Logo o diagnóstico é OBESIDADE SEVERA.
3. Escreva uma função que receba um string contendo uma frase e devolva a quantidade de vogais que foram usadas na frase. Se a entrada for "Ivo viu a uva" a saída deverá ser 7.
4. Uma janela normanda tem o formato de um quadrado sobre o qual se coloca um semicírculo. Escreva uma função que receba o perímetro da janela e indique qual o lado do quadrado no qual ela está inserida. Por exemplo, se o perímetro da janela é de 10m, o lado do quadrado é 2,18m.
5. Escreva uma função que receba uma lista com 3 valores numéricos: A, B e C e calcule as raízes de $Ax^2 + Bx + c = 0$, usando a fórmula de Baskara. Se a entrada for 1, -5 e 6 a resposta deverá ser 2 e 3.
6. Escreva uma função que receba dois números e devolva `True` se tais números puderem indicar um dia e mês e `False` senão. Por exemplo, se a entrada for 23,11 a saída deverá ser `True`, idem se for 1,1. Se a entrada for 33,5 a saída deverá ser `False`.
7. Escreva uma função que receba um número inteiro `x` e calcule e devolva a soma dos múltiplos de 13 menores que `x`. Se a entrada for 100, a saída deverá ser 364 que é igual à soma de 13 26 39 52 65 78 91.
8. Escreva uma função que receba dois números e devolva `True` se tais números puderem indicar uma hora e minuto e `False` senão. Por exemplo, se a entrada for 23,59 a saída deverá ser `True`, idem se for 0,0. Se a entrada for 33,5 a saída deverá ser `False`.
9. Uma empresa de eletricidade cobra de seus clientes uma taxa base de R\$ 10 mensais mais seis centavos por quilowatt-hora para os primeiros 1200 quilowatt-hora e 7 centavos por quilowatt-hora para todo uso acima de 1.200. Escreva uma função que receba o consumo e indique o valor a ser cobrado. Por exemplo, para um consumo de 1000 Kwh, deve cobrar $10+60=70$ R\$.
10. Escreva uma função que receba um parâmetro numérico e retorne a palavra `PAR` ou a palavra `IMPAR` se o parâmetro for par ou ímpar. Por exemplo, se a entrada for 3, a saída deverá ser `IMPAR`.



408-75853 - e /nt

Parâmetros opcionais Seja uma função para sublinhar algo. Poderia ser

```
def sublinha():  
    print("_"*30)
```

Entretanto, nesta função tanto o caracter sublinha (`_`) quanto o valor 30 são fixos. Para permitir alterar isso, pode-se fazer:

```
def sublinha2(n=30, caractere="_"):  
    print(caractere*n)
```

Veja agora como a coisa é chamada

```
>>> def sublinha(n=30, cara="_"):  
    print(cara*n)  
>>> sublinha()
```

```
>>> sublinha(n=10)
```

```
>>> sublinha(cara='$')  
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
```

Funções em Python Além das centenas de funções pré-existentes em Python (print, min, max, sum, len, int, float, input...) a grande vantagem do ambiente é que você pode definir as funções que precisar para o seu trabalho. Depois que uma função foi definida ela passa a se comportar como se fosse (e é) uma função primitiva do Python.

Para definir uma nova função, usa-se o comando def, veja:

```
def nome(parâmetros):  
    comandos
```

Após a palavra def deve vir o nome (único) pelo qual a função será conhecida. Não pode ser uma palavra reservada do Python. Logo depois do nome, é obrigatório um "abre parênteses". A seguir uma quantidade opcional de parâmetros, que são como se fossem nomes de variáveis. Depois desses parâmetros vem o fecha parênteses. Se não houver parâmetro, depois do nome da função vem abre e fecha parênteses. Termina o cabeçalho o caráter "dois pontos" (:). Ele inicia um bloco que limita a definição da função.

Diferentemente do que aconteceu até agora, as linhas que fazem parte da função, não são executadas agora. Elas ficam armazenadas e serão executadas apenas quando a função agora definida, for chamada (executada).

A razão para definir uma função pode ser:

- * Para repetir o mesmo processamento em diversos locais de um programa.
- * Para encapsular um processamento específico.
- * Para comerciar ou apenas trocar algoritmos.
- * Para modularizar um grande processamento, permitindo que o mesmo seja desenvolvido por diversas pessoas.
- * Para poder manusear a complexidade crescente.

A função pode retornar algum resultado para quem a chama. Por exemplo, a função len(lista) retorna um valor que corresponde ao comprimento da lista que é passada como parâmetro.

Veja um exemplo de tudo que se disse

```
def maior(a):  
    i=len(a)-1 #1  
    mai=-99999 #2  
    while i>= 0: #3  
        if a[i]>mai: #4  
            mai=a[i] #5  
            i=i-1 #6  
    return mai #7  
    pass #8  
print(maior([1,5,7,2,3,17,5])) #9
```

Na linha #1 define-se a função de nome maior e se diz que ela deverá ser chamada com um único parâmetro. Quem fez a função sabe que este parâmetro único deve ser uma lista formada por números. Na linha #2 cria-se a variável i e atribui-se a ela o valor do tamanho da lista parâmetro menos 1. Como se sabe tal índice acessa o último elemento da lista. Na linha #3 define-se uma outra variável de controle apta a guardar o maior valor da lista até aquele momento. Como ainda nenhum item foi processado ela vale -∞. Na linha #4 define-se que i vai variar até ter um valor negativo. Na linha #5 se o elemento de ordem i na lista for maior do que a variável mai, mai será atualizada com este novo valor. Na linha #7 que já está fora da indentação do if, mas ainda dentro da indentação do while o índice i é recuado uma posição e graças ao while há um retorno à linha #4. Finalmente, quando o ciclo while acabar, vai-se para a linha #8 que simplesmente devolve o valor calculado a quem chamou a função maior e o processamento desta função está encerrado. A linha #9 não é executada (de qualquer maneira pass não faz nada mesmo), mas lembre-se QUALQUER coisa colocada após um return é meramente decorativo, pois nunca vai ser executado. Na linha #10, já fora da função maior (note a indentação) a mesma vai ser executada. Veja que preciso colocar um print para externalizar o resultado da função. Se não houver print, a função vai ser executada, mas nada será visto de fora.

O parâmetro é uma lista ? Se o parâmetro for uma lista de 4 números, ele será um único parâmetro. Estruturalmente é diferente de 4 parâmetros escalares, que no fundo são uma lista de 4 números. Veja

```
def f1(L4):  
    print(L4[0], L4[1], L4[2], L4[3])  
# ----- ou -----  
def f2(a,b,c,d):  
    print(a, b, c, d)  
f1([1,2,3,4])  
f2(1,2,3,4)
```

Qual estratégia usar ao modularizar um problema e definir quantas e quais funções devem ser construídas é uma arte: a bússola nesta tarefa é a melhor maneira de reaproveitar código e de evitar a reinvenção da roda.

Variáveis locais e globais Como regra geral todas as variáveis definidas dentro de uma função (como i e maior no exemplo aí acima), são chamadas locais à função e isto significa que elas nunca sobrevivem ao tempo de execução da função: não existem antes da função e deixam de existir quando a função acaba. Desta maneira não podem ter seu valor consultado e/ou alterado por alguém externo à função.

Este fato permite que duas funções distintas tenham variáveis com o mesmo nome (como é muito comum um índice chamado i) sem que um atrapalhe o outro. Desta maneira a única forma de ceder e receber valores entre funções é através de parâmetros e de retornos.

Já uma variável que precise ser vista e alterada por outros é conhecida como variável GLOBAL. Ela deve ser definida dentro de um módulo, mas FORA de uma função. Agora ela pode ser manuseada por todas as funções desse módulo e por todos os módulos que importam o módulo que a definiu.

Para que uma função possa alterar uma variável global, a mesma deve ser definida como global usando-se o comando GLOBAL como linha seguinte à definição.

Veja um exemplo:

```
xu = 77  
def fun():  
    xu = 2  
    print(xu)  
print(xu) # 77 é impresso  
fun() # 2 é impresso  
print(xu) # 77 é impresso  
def alva():  
    global xu  
    xu=2  
alva()  
print(xu) # 2 é impresso
```

As boas práticas de programação sugerem não usar variáveis globais. Esta prática viola o encapsulamento e é fonte frequente de bugs de programação.

Seus módulos podem ser importados Organizando seu trabalho direitinho, você pode modularizar seus programas e importá-los quando necessário, acompanhe

```
#----- beta.py -----  
def alfa(x):  
    return x**3  
#-----  
>>> import beta  
>>> print(beta.alfa(4))  
#-----  
>>> from beta import *  
>>> print(alfa(4))
```

Para você fazer

A seguir uma lista de funções que você deve programar em Python. Imprima as funções e devolva-as impressas junto com esta folha. Não esqueça de identificar com seu nome e o código 9 limite entrega 10/10/24

1. Escreva uma função que receba um número inteiro x e devolva True se ele for primo e False senão. Um número é primo quando seus únicos divisores são ele mesmo e a unidade. Se a entrada for 17 a saída deverá ser True.

2. Um plano de telefonia celular tem uma taxa de R\$ 35 por mês. O plano inclui 400 minutos gratuitos e taxa de 10 centavos para cada minuto adicional utilizado. Escreva uma função que indique o custo mensal em função do número de minutos utilizados. Por exemplo se a função for chamada com 410, deve responder 36 R\$.
3. Escreva uma função que receba um parâmetro numérico e retorne a palavra PAR ou a palavra IMPAR se o parâmetro for par ou ímpar. Por exemplo, se a entrada for 3, a saída deverá ser IMPAR.
4. Escreva uma função que receba um número inteiro e devolva uma lista com os divisores deste número. Se a entrada for 16, a saída deverá ser 1,2,4,8,16.
5. Escreva uma função que receba um string contendo uma frase e devolva a quantidade de palavras que formam a frase. Se a entrada for "IVO VIU A UVA" a saída deverá ser 4.
6. Escreva uma função que receba uma lista formada por 3 números (se vierem menos ou mais deve dar um erro e encerrar) e deve imprimir SIM se os números forem lados de um triângulo retângulo e NÃO se não puderem ser. Por exemplo, se a entrada for 3,5,4 a saída deverá ser SIM. Se a entrada for 1,10,100 a saída deverá ser NÃO.
7. Escreva uma função que receba uma lista de números e devolva o menor deles. Se a entrada for 33, 22, -10, -4 e 33 a saída deve ser -10.
8. Escreva uma função que receba uma lista de números contendo uma distribuição qualquer e devolva a distribuição acumulada. Por exemplo, se a entrada for 1, 5, 7, 10, 21 a saída deverá ser 1, 6, 13, 23, 44.
9. Escreva uma função que receba um string contendo uma frase e devolva a quantidade de palavras masculinas que formam a frase. Por hipótese, as palavras masculinas são as que terminam em "O" ou "o". Se a entrada for "O RATO ROEU A ROUPA DO REI" a saída deverá ser 3.
10. Em uma certa província a velocidade máxima permitida em estradas é de 100 km por hora e a velocidade mínima é de 50 km por hora. A multa por violar estes limites é de R\$ 10 para cada km por hora acima da velocidade máxima ou abaixo da velocidade mínima. Escreva uma função que receba a velocidade de condução e indique quanto deve ser a multa se é que ela deva existir. Em caso negativo, retornar 0. Por exemplo, se for chamada com 75 km/h, deve retornar 0. Se for chamada com 110 ou com 40 Km/h deve retornar 100,00.



408-75860 - e /nt

Parâmetros opcionais Seja uma função para sublinhar algo. Poderia ser

```
def sublinha():  
    print("_"*30)  
Entretanto, nesta função tanto o caracter sublinha (_) quanto o valor 30 são fixos. Para permitir alterar isso, pode-se fazer:  
def sublinha2(n=30, caractere="_"):  
    print(caractere*n)  
Veja agora como a coisa é chamada  
>>> def sublinha(n=30, cara="_"):  
    print(cara*n)  
>>> sublinha()  
-----  
>>> sublinha(n=10)  
-----  
>>> sublinha(cara='$')  
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
```

Funções em Python Além das centenas de funções pré-existentes em Python (print, min, max, sum, len, int, float, input...) a grande vantagem do ambiente é que você pode definir as funções que precisar para o seu trabalho. Depois que uma função foi definida ela passa a se comportar como se fosse (e é) uma função primitiva do Python.

Para definir uma nova função, usa-se o comando def, veja:

```
def nome(parâmetros):  
    comandos
```

Após a palavra def deve vir o nome (único) pelo qual a função será conhecida. Não pode ser uma palavra reservada do Python. Logo depois do nome, é obrigatório um "abre parênteses". A seguir uma quantidade opcional de parâmetros, que são como se fossem nomes de variáveis. Depois desses parâmetros vem o fecha parênteses. Se não houver parâmetro, depois do nome da função vem abre e fecha parênteses. Termina o cabeçalho o caráter "dois pontos" (:). Ele inicia um bloco que limita a definição da função.

Diferentemente do que aconteceu até agora, as linhas que fazem parte da função, não são executadas agora. Elas ficam armazenadas e serão executadas apenas quando a função agora definida, for chamada (executada).

A razão para definir uma função pode ser:

- * Para repetir o mesmo processamento em diversos locais de um programa.
- * Para encapsular um processamento específico.
- * Para comercializar ou apenas trocar algoritmos.
- * Para modularizar um grande processamento, permitindo que o mesmo seja desenvolvido por diversas pessoas.
- * Para poder manusear a complexidade crescente.

A função pode retornar algum resultado para quem a chama. Por exemplo, a função len(lista) retorna um valor que corresponde ao comprimento da lista que é passada como parâmetro.

Veja um exemplo de tudo que se disse

```
def maior(a):  
    i=len(a)-1 #1  
    mai=-99999 #2  
    while i>= 0: #3  
        if a[i]>mai: #4  
            mai=a[i] #5  
            i=i-1 #6  
    return mai #7  
    pass #8  
print(maior([1,5,7,2,3,17,5])) #9
```

Na linha #1 define-se a função de nome maior e se diz que ela deverá ser chamada com um único parâmetro. Quem fez a função sabe que este parâmetro único deve ser uma lista formada por números. Na linha #2 cria-se a variável i e atribui-se a ela o valor do tamanho da lista parâmetro menos 1. Como se sabe tal índice acessa o último elemento da lista. Na linha #3 define-se uma outra variável de controle apta a guardar o maior valor da lista até aquele momento. Como ainda nenhum item foi processado ela vale -∞. Na linha #4 define-se que i vai variar até ter um valor negativo. Na linha #5 se o elemento de ordem i na lista for maior do que a variável mai, mai será atualizada com este novo valor. Na linha #7 que já está fora da indentação do if, mas ainda dentro da indentação do while o índice i é recuado uma posição e graças ao while há um retorno à linha #4. Finalmente, quando o ciclo while acabar, vai-se para a linha #8 que simplesmente devolve o valor calculado a quem chamou a função maior e o processamento desta função está encerrado. A linha #9 não é executada (de qualquer maneira pass não faz nada mesmo), mas lembre-se QUALQUER coisa colocada após um return é meramente decorativo, pois nunca vai ser executado. Na linha #10, já fora da função maior (note a indentação) a mesma vai ser executada. Veja que preciso colocar um print para externalizar o resultado da função. Se não houver print, a função vai ser executada, mas nada será visto de fora.

O parâmetro é uma lista ? Se o parâmetro for uma lista de 4 números, ele será um único parâmetro. Estruturalmente é diferente de 4 parâmetros escalares, que no fundo são uma lista de 4 números. Veja

```
def f1(L4):  
    print(L4[0], L4[1], L4[2], L4[3])  
# ----- ou -----  
def f2(a,b,c,d):  
    print(a, b, c, d)  
f1([1,2,3,4])  
f2(1,2,3,4)
```

Qual estratégia usar ao modularizar um problema e definir quantas e quais funções devem ser construídas é uma arte: a bússola nesta tarefa é a melhor maneira de reaproveitar código e de evitar a reinvenção da roda.

Variáveis locais e globais Como regra geral todas as variáveis definidas dentro de uma função (como i e maior no exemplo aí acima), são chamadas locais à função e isto significa que elas nunca sobrevivem ao tempo de execução da função: não existem antes da função e deixam de existir quando a função acaba. Desta maneira não podem ter seu valor consultado e/ou alterado por alguém externo à função.

Este fato permite que duas funções distintas tenham variáveis com o mesmo nome (como é muito comum um índice chamado i) sem que um atrapalhe o outro. Desta maneira a única forma de ceder e receber valores entre funções é através de parâmetros e de retornos.

Já uma variável que precise ser vista e alterada por outros é conhecida como variável GLOBAL. Ela deve ser definida dentro de um módulo, mas FORA de uma função. Agora ela pode ser manuseada por todas as funções desse módulo e por todos os módulos que importam o módulo que a definiu.

Para que uma função possa alterar uma variável global, a mesma deve ser definida como global usando-se o comando GLOBAL como linha seguinte à definição.

Veja um exemplo:

```
xu = 77  
def fun():  
    xu = 2  
    print(xu)  
print(xu) # 77 é impresso  
fun() # 2 é impresso  
print(xu) # 77 é impresso  
def alva():  
    global xu  
    xu=2  
alva()  
print(xu) # 2 é impresso
```

As boas práticas de programação sugerem não usar variáveis globais. Esta prática viola o encapsulamento e é fonte frequente de bugs de programação.

Seus módulos podem ser importados Organizando seu trabalho direitinho, você pode modularizar seus programas e importá-los quando necessário, acompanhe

```
#----- beta.py -----  
def alfa(x):  
    return x**3  
#-----  
>>> import beta  
>>> print(beta.alfa(4))  
#-----  
>>> from beta import *  
>>> print(alfa(4))
```

Para você fazer

A seguir uma lista de funções que você deve programar em Python. Imprima as funções e devolva-as impressas junto com esta folha. Não esqueça de identificar com seu nome e o código 10 limite entrega 10/10/24

1. Escreva uma função que receba um número inteiro x e devolva True se ele for primo e False senão. Um número é primo quando seus únicos divisores são ele mesmo e a unidade. Se a entrada for 17 a saída deverá ser True.

2. Um plano de telefonia celular tem uma taxa de R\$ 35 por mês. O plano inclui 400 minutos gratuitos e taxa de 10 centavos para cada minuto adicional utilizado. Escreva uma função que indique o custo mensal em função do número de minutos utilizados. Por exemplo se a função for chamada com 410, deve responder 36 R\$.
3. Escreva uma função que receba um número inteiro e devolva uma lista com os divisores deste número. Se a entrada for 16, a saída deverá ser 1,2,4,8,16.
4. Escreva uma função que receba o valor de x e calcule e devolva o valor de $S = 1/4 + 2/9 + 3/16 + \dots + x/(x+1)^2$. Se a entrada for 5 a saída deverá ser 0.958611111.
5. Escreva uma função que receba uma lista de números positivos e negativos misturados e devolva duas listas: a primeira contendo apenas os positivos e a segunda apenas os negativos, na mesma ordem em que eles apareceram na entrada. Se a entrada for -2, 4, -4, -6, -11, 4, 18 as saídas deverão ser -2, -4, -6, -11 e 4, 4, 18.
6. Escreva uma função que receba o valor de x e calcule e devolva o valor de $S = 1/2 + 2/3 + 3/4 + \dots + x/x + 1$. Se a entrada for 5 a saída deverá ser 3.55.
7. Escreva uma função que receba uma lista com 3 valores numéricos: A, B e C e calcule as raízes de $Ax^2 + Bx + c = 0$, usando a fórmula de Baskara. Se a entrada for 1, -5 e 6 a resposta deverá ser 2 e 3.
8. Uma janela normanda tem o formato de um quadrado sobre o qual se coloca um semicírculo. Escreva uma função que receba o perímetro da janela e indique qual o lado do quadrado no qual ela está inserida. Por exemplo, se o perímetro da janela é de 10m, o lado do quadrado é 2,18m.
9. Escreva uma função que receba a altura e o peso de uma pessoa. Deve calcular o IMC - índice de massa corporal ($IMV = \text{peso}/\text{altura}^2$) medidos em kg e m. A seguir deve emitir mensagem a depender do IMC. $IMC < 25$ =adequado; $IMC < 30$ =sobrepeso; $IMC < 35$ =obesidade e $IMC \geq 35$ =obesidade severa. Por exemplo, se o sujeito pesa 110Kg e tem 1.70m de altura seu IMC é de 38.06. Logo o diagnóstico é OBESIDADE SEVERA.
10. Escreva uma função que receba dois números: o preço do álcool e o preço da gasolina. Sabendo que um carro em particular faz um certo trajeto gastando 10l de álcool e 7l de gasolina (o mesmo trajeto), a função deve recomendar o combustível mais barato, a saber ALCOOL ou GASOLINA. Por exemplo, se A=2.90 R\$ e G=4.10 R\$, fazendo as contas, a gasolina é mais eficiente e a função deve recomendá-la.



408-75877 - e /nt

Parâmetros opcionais Seja uma função para sublinhar algo. Poderia ser

```
def sublinha():  
    print("_"*30)  
Entretanto, nesta função tanto o caracter sublinha (_ ) quanto o valor 30 são fixos. Para permitir alterar isso, pode-se fazer:  
def sublinha2(n=30, caractere="_"):  
    print(caractere*n)  
Veja agora como a coisa é chamada  
>>> def sublinha(n=30, cara="_"):  
    print(cara*n)  
>>> sublinha()
```

```
>>> sublinha(n=10)  
-----  
>>> sublinha(cara='$')  
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
```

Funções em Python Além das centenas de funções pré-existentes em Python (print, min, max, sum, len, int, float, input...) a grande vantagem do ambiente é que você pode definir as funções que precisar para o seu trabalho. Depois que uma função foi definida ela passa a se comportar como se fosse (e é) uma função primitiva do Python.

Para definir uma nova função, usa-se o comando `def`, veja:

```
def nome(parâmetros):  
    comandos
```

Após a palavra `def` deve vir o nome (único) pelo qual a função será conhecida. Não pode ser uma palavra reservada do Python. Logo depois do nome, é obrigatório um “abre parênteses”. A seguir uma quantidade opcional de parâmetros, que são como se fossem nomes de variáveis. Depois desses parâmetros vem o fecha parênteses. Se não houver parâmetro, depois do nome da função vem abre e fecha parênteses. Termina o cabeçalho o caractere “dois pontos” (:). Ele inicia um bloco que limita a definição da função.

Diferentemente do que aconteceu até agora, as linhas que fazem parte da função, não são executadas agora. Elas ficam armazenadas e serão executadas apenas quando a função agora definida, for chamada (executada).

A razão para definir uma função pode ser:

- * Para repetir o mesmo processamento em diversos locais de um programa.
- * Para encapsular um processamento específico.
- * Para comerciar ou apenas trocar algoritmos.
- * Para modularizar um grande processamento, permitindo que o mesmo seja desenvolvido por diversas pessoas.
- * Para poder manusear a complexidade crescente.

A função pode retornar algum resultado para quem a chama. Por exemplo, a função `len(lista)` retorna um valor que corresponde ao comprimento da lista que é passada como parâmetro.

Veja um exemplo de tudo que se disse

```
def maior(a):  
    i=len(a)-1 #1  
    mai=-99999 #2  
    while i>= 0: #3  
        if a[i]>mai: #4  
            mai=a[i] #5  
            i=i-1 #6  
    return mai #7  
    pass #8  
print(maior([1,5,7,2,3,17,5])) #9
```

Na linha #1 define-se a função de nome `maior` e se diz que ela deverá ser chamada com um único parâmetro. Quem fez a função sabe que este parâmetro único deve ser uma lista formada por números. Na linha #2 cria-se a variável `i` e atribui-se a ela o valor do tamanho da lista parâmetro menos 1. Como se sabe tal índice acessa o último elemento da lista. Na linha #3 define-se uma outra variável de controle `apt` a guardar o maior valor da lista até aquele momento. Como ainda nenhum item foi processado ela vale $-\infty$. Na linha #4 define-se que `i` vai variar até ter um valor negativo. Na linha #5 se o elemento de ordem `i` na lista for maior do que a variável `mai`, `mai` será atualizada com este novo valor. Na linha #7 que já está fora da indentação do `if`, mas ainda dentro da indentação do `while` o índice `i` é recuado uma posição e graças ao `while` há um retorno à linha #4. Finalmente, quando o ciclo `while` acabar, vai-se para a linha #8 que simplesmente devolve o valor calculado a quem chamou a função `maior` e o processamento desta função está encerrado. A linha #9 não é executada (de qualquer maneira `pass` não faz nada mesmo), mas lembre-se QUALQUER coisa colocada após um `return` é meramente decorativo, pois nunca vai ser executado. Na linha #10, já fora da função `maior` (note a indentação) a mesma vai ser executada. Veja que preciso colocar um `print` para externalizar o resultado da função. Se não houver `print`, a função vai ser executada, mas nada será visto de fora.

O parâmetro é uma lista ? Se o parâmetro for uma lista de 4 números, ele será um único parâmetro. Estruturalmente é diferente de 4 parâmetros escalares, que no fundo são uma lista de 4 números. Veja

```
def f1(L4):  
    print(L4[0], L4[1], L4[2], L4[3])  
# ----- OU -----  
def f2(a,b,c,d):  
    print(a, b, c, d)  
f1([1,2,3,4])  
f2(1,2,3,4)
```

Qual estratégia usar ao modularizar um problema e definir quantas e quais funções devem ser construídas é uma arte: a bússola nesta tarefa é a melhor maneira de reaproveitar código e de evitar a reinvenção da roada.

Variáveis locais e globais Como regra geral todas as variáveis definidas dentro de uma função (como `i` e `maior` no exemplo aí acima), são chamadas **locais** à função e isto significa que elas nunca sobrevivem ao tempo de execução da função: não existem antes da função e deixam de existir quando a função acaba. Desta maneira não podem ter seu valor consultado e/ou alterado por alguém externo à função.

Este fato permite que duas funções distintas tenham variáveis com o mesmo nome (como é muito comum um índice chamado `i`) sem que um atrapalhe o outro. Desta maneira a única forma de ceder e receber valores entre funções é através de parâmetros e de retornos.

Já uma variável que precise ser vista e alterada por outros é conhecida como variável **GLOBAL**. Ela deve ser definida dentro de um módulo, mas **FORA** de uma função. Agora ela pode ser manuseada por todas as funções desse módulo e por todos os módulos que importam o módulo que a definiu.

Para que uma função possa alterar uma variável global, a mesma deve ser definida como global usando-se o comando `GLOBAL` como linha seguinte à definição.

Veja um exemplo:

```
xu = 77  
def fun():  
    xu = 2  
    print(xu)  
print(xu) # 77 é impresso  
fun() # 2 é impresso  
print(xu) # 77 é impresso  
def alva():  
    global xu  
    xu=2  
alva()  
print(xu) # 2 é impresso
```

As boas práticas de programação sugerem não usar variáveis globais. Esta prática viola o encapsulamento e é fonte frequente de bugs de programação.

Seus módulos podem ser importados Organizando seu trabalho direitinho, você pode modularizar seus programas e importá-los quando necessário, acompanhe

```
#----- beta.py -----  
def alfa(x):  
    return x**3  
#-----  
>>> import beta  
>>> print(beta.alfa(4))  
#-----  
>>> from beta import *  
>>> print(alfa(4))
```

☞ Para você fazer

A seguir uma lista de funções que você deve programar em Python. Imprima as funções e devolva-as impressas junto com esta folha. Não esqueça de identificar com seu nome e o código 11 limite entrega 10/10/24

1. Escreva uma função que receba uma lista de números calcule e devolva a média dos elementos da lista. Se a entrada for 8 7 4 19 21 a saída deverá ser 11.8.
2. Escreva uma função que receba o valor de x e calcule e devolva o valor de $S = 1/2 + 2/3 + 3/4 + \dots + x/x + 1$. Se a entrada for 5 a saída deverá ser 3.55.

3. Escreva uma função que receba um número inteiro e devolva uma lista com os divisores deste número. Se a entrada for 16, a saída deverá ser 1,2,4,8,16.
4. Escreva uma função que receba uma lista de números positivos e negativos misturados e devolva duas listas: a primeira contendo apenas os positivos e a segunda apenas os negativos, na mesma ordem em que eles apareceram na entrada. Se a entrada for -2, 4, -4, -6, -11, 4, 18 as saídas deverão ser -2, -4, -6, -11 e 4, 4, 18.
5. Em uma certa província a velocidade máxima permitida em estradas é de 100 km por hora e a velocidade mínima é de 50 km por hora. A multa por violar estes limites é de R\$ 10 para cada km por hora acima da velocidade máxima ou abaixo da velocidade mínima. Escreva uma função que receba a velocidade de condução e indique quanto deve ser a multa se é que ela deva existir. Em caso negativo, retornar 0. Por exemplo, se for chamada com 75 km/h, deve retornar 0. Se for chamada com 110 ou com 40 Km/h deve retornar 100,00.
6. Escreva uma função que receba um número inteiro e devolva a soma de todos os divisores deste número. Se a entrada for 15 a saída deverá ser 24.
7. Escreva uma função que receba 2 valores: o peso e a altura de alguém e devolva o IMC (Índice de Massa Corporal): $IMC = \text{peso} \div (\text{altura})^2$. Peso em Kg e altura em m. Se a entrada for 80, 1.80 a saída deverá ser 24.69.
8. Um plano de telefonia celular tem uma taxa de R\$ 35 por mês. O plano inclui 400 minutos gratuitos e taxa de 10 centavos para cada minuto adicional utilizado. Escreva uma função que indique o custo mensal em função do número de minutos utilizados. Por exemplo se a função for chamada com 410, deve responder 36 R\$.
9. Escreva uma função que receba 2 horários (no formato HH, MM) indicando o início e o fim de uma determinada tarefa. A função deve devolver a duração em minutos da tarefa. Se a entrada for 1,10 e 2,0 a saída deverá ser 50.
10. Uma empresa de eletricidade cobra de seus clientes uma taxa base de R\$ 10 mensais mais seis centavos por quilowatt-hora para os primeiros 1200 quilowatt-hora e 7 centavos por quilowatt-hora para todo uso acima de 1.200. Escreva uma função que receba o consumo e indique o valor a ser cobrado. Por exemplo, para um consumo de 1000 Kwh, deve cobrar 10+60=70 R\$.



408-75884 - e / nt

Parâmetros opcionais Seja uma função para sublinhar algo. Poderia ser

```
def sublinha():  
    print("_"*30)  
Entretanto, nesta função tanto o caracter sublinha ( _ ) quanto o valor 30 são fixos. Para permitir alterar isso, pode-se fazer:  
def sublinha2(n=30, caractere="_"):  
    print(caractere*n)  
Veja agora como a coisa é chamada  
>>> def sublinha(n=30, cara="_"):  
    print(cara*n)  
>>> sublinha()  
-----  
>>> sublinha(n=10)  
-----  
>>> sublinha(cara='$')  
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
```

Funções em Python Além das centenas de funções pré-existentes em Python (print, min, max, sum, len, int, float, input...) a grande vantagem do ambiente é que você pode definir as funções que precisar para o seu trabalho. Depois que uma função foi definida ela passa a se comportar como se fosse (e é) uma função primitiva do Python.

Para definir uma nova função, usa-se o comando `def`, veja:

```
def nome(parâmetros):  
    comandos
```

Após a palavra `def` deve vir o nome (único) pelo qual a função será conhecida. Não pode ser uma palavra reservada do Python. Logo depois do nome, é obrigatório um "abre parênteses". A seguir uma quantidade opcional de parâmetros, que são como se fossem nomes de variáveis. Depois desses parâmetros vem o fecha parênteses. Se não houver parâmetro, depois do nome da função vem abre e fecha parênteses. Termina o cabeçalho o caráter "dois pontos" (:). Ele inicia um bloco que limita a definição da função.

Diferentemente do que aconteceu até agora, as linhas que fazem parte da função, não são executadas agora. Elas ficam armazenadas e serão executadas apenas quando a função agora definida, for chamada (executada).

A razão para definir uma função pode ser:

- * Para repetir o mesmo processamento em diversos locais de um programa.
- * Para encapsular um processamento específico.
- * Para comerciar ou apenas trocar algoritmos.
- * Para modularizar um grande processamento, permitindo que o mesmo seja desenvolvido por diversas pessoas.
- * Para poder manusear a complexidade crescente.

A função pode retornar algum resultado para quem a chama. Por exemplo, a função `len(lista)` retorna um valor que corresponde ao comprimento da lista que é passada como parâmetro.

Veja um exemplo de tudo que se disse

```
def maior(a):  
    #1  
    i=len(a)-1 #2  
    mai=-99999 #3  
    while i>= 0:  
        #4  
        if a[i]>mai:  
            #5  
            mai=a[i] #6  
            i=i-1 #7  
    return mai #8  
    pass #9  
print(maior([1,5,7,2,3,17,5])) #10
```

Na linha #1 define-se a função de nome `maior` e se diz que ela deverá ser chamada com um único parâmetro. Quem fez a função sabe que este parâmetro único deve ser uma lista formada por números. Na linha #2 cria-se a variável `i` e atribui-se a ela o valor do tamanho da lista parâmetro menos 1. Como se sabe tal índice acessa o último elemento da lista. Na linha #3 define-se uma outra variável de controle `apt` a guardar o maior valor da lista até aquele momento. Como ainda nenhum item foi processado ela vale $-\infty$. Na linha #4 define-se que `i` vai variar até ter um valor negativo. Na linha #5 se o elemento de ordem `i` na lista for maior do que a variável `mai`, `mai` será atualizada com este novo valor. Na linha #7 que já está fora da indentação do `if`, mas ainda dentro da indentação do `while` o índice `i` é recuado uma posição e graças ao `while` há um retorno à linha #4. Finalmente, quando o ciclo `while` acabar, vai-se para a linha #8 que simplesmente devolve o valor calculado a quem chamou a função `maior` e o processamento desta função está encerrado. A linha #9 não é executada (de qualquer maneira `pass` não faz nada mesmo), mas lembre-se QUALQUER coisa colocada após um `return` é meramente decorativo, pois nunca vai ser executado. Na linha #10, já fora da função `maior` (note a indentação) a mesma vai ser executada. Veja que preciso colocar um `print` para externalizar o resultado da função. Se não houver `print`, a função vai ser executada, mas nada será visto de fora.

O parâmetro é uma lista ? Se o parâmetro for uma lista de 4 números, ele será um único parâmetro. Estruturalmente é diferente de 4 parâmetros escalares, que no fundo são uma lista de 4 números. Veja

```
def f1(L4):  
    print(L4[0], L4[1], L4[2], L4[3])  
    # ---- 00 ----  
def f2(a,b,c,d):  
    print(a, b, c, d)  
f1([1,2,3,4])  
f2(1,2,3,4)
```

Qual estratégia usar ao modularizar um problema e definir quantas e quais funções devem ser construídas é uma arte: a bússola nesta tarefa é a melhor maneira de reaproveitar código e de evitar a reinvenção da roda.

Variáveis locais e globais Como regra geral todas as variáveis definidas dentro de uma função (como `i` e `maior` no exemplo aí acima), são chamadas **locais** à função e isto significa que elas nunca sobrevivem ao tempo de execução da função: não existem antes da função e deixam de existir quando a função acaba. Desta maneira não podem ter seu valor consultado e/ou alterado por alguém externo à função.

Este fato permite que duas funções distintas tenham variáveis com o mesmo nome (como é muito comum um índice chamado `i`) sem que um atrapalhe o outro. Desta maneira a única forma de ceder e receber valores entre funções é através de parâmetros e de retornos.

Já uma variável que precise ser vista e alterada por outros é conhecida como variável **GLOBAL**. Ela deve ser definida dentro de um módulo, mas **FORA** de uma função. Agora ela pode ser manuseada por todas as funções desse módulo e por todos os módulos que importam o módulo que a definiu.

Para que uma função possa alterar uma variável global, a mesma deve ser definida como global usando-se o comando `GLOBAL` como linha seguinte à definição.

Veja um exemplo:

```
xu = 77  
def fun():  
    xu = 2  
    print(xu)  
print(xu) # 77 é impresso  
fun() # 2 é impresso  
print(xu) # 77 é impresso  
def alva():  
    global xu  
    xu=2  
alva()  
print(xu) # 2 é impresso
```

As boas práticas de programação sugerem não usar variáveis globais. Esta prática viola o encapsulamento e é fonte frequente de bugs de programação.

Seus módulos podem ser importados Organizando seu trabalho direitinho, você pode modularizar seus programas e importá-los quando necessário, acompanhe

```
#----- beta.py -----  
def alfa(x):  
    return x**3  
#-----  
>>> import beta  
>>> print(beta.alfa(4))  
#-----  
>>> from beta import *  
>>> print(alfa(4))
```

☞ Para você fazer

A seguir uma lista de funções que você deve programar em Python. Imprima as funções e devolva-as impressas junto com esta folha. Não esqueça de identificar com seu nome e o código 12 limite entrega 10/10/24

1. Escreva uma função que receba um número inteiro `x` e devolva `True` se ele for primo e `False` senão. Um número é primo quando seus únicos divisores são ele mesmo e a unidade. Se a entrada for 17 a saída deverá ser `True`.

2. Escreva uma função que receba 2 horários (no formato HH, MM) indicando o início e o fim de uma determinada tarefa. A função deve devolver a duração em minutos da tarefa. Se a entrada for 1,10 e 2,0 a saída deverá ser 50.
3. Escreva uma função que receba um string contendo uma frase e devolva a quantidade de palavras que formam a frase. Se a entrada for "IVO VIU A UVA" a saída deverá ser 4.
4. Escreva uma função que receba uma lista de números e imprima-a em ordem reversa (de trás para a frente). Se a entrada for 4, 7, -3, 22, 14, a impressão deverá ser de 14, 22, -3, 7 e 4.
5. Escreva uma função que receba um string contendo uma frase e devolva a quantidade de palavras masculinas que formam a frase. Por hipótese, as palavras masculinas são as que terminam em "O" ou "o". Se a entrada for "O RATO ROEU A ROUPA DO REI" a saída deverá ser 3.
6. Em uma certa província a velocidade máxima permitida em estradas é de 100 km por hora e a velocidade mínima é de 50 km por hora. A multa por violar estes limites é de R\$ 10 para cada km por hora acima da velocidade máxima ou abaixo da velocidade mínima. Escreva uma função que receba a velocidade de condução e indique quanto deve ser a multa se é que ela deva existir. Em caso negativo, retornar 0. Por exemplo, se for chamada com 75 km/h, deve retornar 0. Se for chamada com 110 ou com 40 Km/h deve retornar 100,00.
7. Escreva uma função que receba um parâmetro numérico e retorne a palavra `PAR` ou a palavra `IMPAR` se o parâmetro for par ou ímpar. Por exemplo, se a entrada for 3, a saída deverá ser `IMPAR`.
8. Escreva uma função que receba dois números que descrevem o desempenho de alguém ao cursar a disciplina de **MÉTODOS NUMÉRICOS**. O primeiro número é a frequência ($0.0 \leq f \leq 100.0$) em percentagem e o segundo número é a nota final obtida ($0.0 \leq NF \leq 10.0$). Deve imprimir a mensagem `APROVADO` ou `REPROVADO` a depender das regras da UTFPR para isso. Por exemplo, se $f = 66$ e $NF = 7.1$ a mensagem deve ser `REPROVADO`.
9. Escreva uma função que receba um valor numérico indicando o lado de um quadrado, medido em metros e devolva o valor do perímetro e da área do quadrado. Se a entrada for 1, a saída deve ser 4 e 1.
10. Escreva uma função que receba uma lista de números e devolva o maior deles. Se a entrada for 33, 22, -10, -4 e 33 a saída deve ser 33.



408-75989 - e /nt

Parâmetros opcionais Seja uma função para sublinhar algo. Poderia ser

```
def sublinha():  
    print("_"*30)  
Entretanto, nesta função tanto o caracter sublinha ( _ ) quanto o valor 30 são fixos. Para permitir alterar isso, pode-se fazer:  
def sublinha2(n=30, caractere="_"):  
    print(caractere*n)  
Veja agora como a coisa é chamada  
>>> def sublinha(n=30, cara="_"):  
    print(cara*n)  
>>> sublinha()  
-----  
>>> sublinha(n=10)
```

```
-----  
>>> sublinha(cara='$')  
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
```

Funções em Python Além das centenas de funções pré-existentes em Python (print, min, max, sum, len, int, float, input...) a grande vantagem do ambiente é que você pode definir as funções que precisar para o seu trabalho. Depois que uma função foi definida ela passa a se comportar como se fosse (e é) uma função primitiva do Python.

Para definir uma nova função, usa-se o comando `def`, veja:

```
def nome(parâmetros):  
    comandos
```

Após a palavra `def` deve vir o nome (único) pelo qual a função será conhecida. Não pode ser uma palavra reservada do Python. Logo depois do nome, é obrigatório um “abre parênteses”. A seguir uma quantidade opcional de parâmetros, que são como se fossem nomes de variáveis. Depois desses parâmetros vem o fecha parênteses. Se não houver parâmetro, depois do nome da função vem abre e fecha parênteses. Termina o cabeçalho o caráter “dois pontos” (:). Ele inicia um bloco que limita a definição da função.

Diferentemente do que aconteceu até agora, as linhas que fazem parte da função, não são executadas agora. Elas ficam armazenadas e serão executadas apenas quando a função agora definida, for chamada (executada).

A razão para definir uma função pode ser:

- * Para repetir o mesmo processamento em diversos locais de um programa.
- * Para encapsular um processamento específico.
- * Para comercializar ou apenas trocar algoritmos.
- * Para modularizar um grande processamento, permitindo que o mesmo seja desenvolvido por diversas pessoas.
- * Para poder manusear a complexidade crescente.

A função pode retornar algum resultado para quem a chama. Por exemplo, a função `len(lista)` retorna um valor que corresponde ao comprimento da lista que é passada como parâmetro.

Veja um exemplo de tudo que se disse

```
def maior(a):  
    #1          #1  
    i=len(a)-1  #2  
    mai=-99999 #3  
    while i>= 0:  
        #4  
        if a[i]>mai:  
            #5  
            mai=a[i] #6  
        i=i-1 #7  
    return mai #8  
    pass #9  
print(maior([1,5,7,2,3,17,5])) #10
```

Na linha #1 define-se a função de nome `maior` e se diz que ela deverá ser chamada com um único parâmetro. Quem fez a função sabe que este parâmetro único deve ser uma lista formada por números. Na linha #2 cria-se a variável `i` e atribui-se a ela o valor do tamanho da lista parâmetro menos 1. Como se sabe tal índice acessa o último elemento da lista. Na linha #3 define-se uma outra variável de controle apta a guardar o maior valor da lista até aquele momento. Como ainda nenhum item foi processado ela vale $-\infty$. Na linha #4 define-se que `i` vai variar até ter um valor negativo. Na linha #5 se o elemento de ordem `i` na lista for maior do que a variável `mai`, `mai` será atualizada com este novo valor. Na linha #7 que já está fora da indentação do `if`, mas ainda dentro da indentação do `while` o índice `i` é recuado uma posição e graças ao `while` há um retorno à linha #4. Finalmente, quando o ciclo `while` acabar, vai-se para a linha #8 que simplesmente devolve o valor calculado a quem chamou a função `maior` e o processamento desta função está encerrado. A linha #9 não é executada (de qualquer maneira `pass` não faz nada mesmo), mas lembre-se QUALQUER coisa colocada após um `return` é meramente decorativo, pois nunca vai ser executado. Na linha #10, já fora da função `maior` (note a indentação) a mesma vai ser executada. Veja que preciso colocar um `print` para externalizar o resultado da função. Se não houver `print`, a função vai ser executada, mas nada será visto de fora.

O parâmetro é uma lista ? Se o parâmetro for uma lista de 4 números, ele será um único parâmetro. Estruturalmente é diferente de 4 parâmetros escalares, que no fundo são uma lista de 4 números. Veja

```
def f1(L4):  
    print(L4[0], L4[1], L4[2], L4[3])  
# ----- OU -----  
def f2(a,b,c,d):  
    print(a, b, c, d)  
f1([1,2,3,4])  
f2(1,2,3,4)
```

Qual estratégia usar ao modularizar um problema e definir quantas e quais funções devem ser construídas é uma arte: a bússola nesta tarefa é a melhor maneira de reaproveitar código e de evitar a reinvenção da roda.

Variáveis locais e globais Como regra geral todas as variáveis definidas dentro de uma função (como `i` e `maior` no exemplo aí acima), são chamadas **locais** à função e isto significa que elas nunca sobrevivem ao tempo de execução da função: não existem antes da função e deixam de existir quando a função acaba. Desta maneira não podem ter seu valor consultado e/ou alterado por alguém externo à função.

Este fato permite que duas funções distintas tenham variáveis com o mesmo nome (como é muito comum um índice chamado `i`) sem que um atrapalhe o outro. Desta maneira a única forma de ceder e receber valores entre funções é através de parâmetros e de retornos.

Já uma variável que precise ser vista e alterada por outros é conhecida como variável **GLOBAIS**. Ela deve ser definida dentro de um módulo, mas **FORA** de uma função. Agora ela pode ser manuseada por todas as funções desse módulo e por todos os módulos que importam o módulo que a definiu.

Para que uma função possa alterar uma variável global, a mesma deve ser definida como global usando-se o comando `GLOBAL` como linha seguinte à definição.

Veja um exemplo:

```
xu = 77  
def fun():  
    xu = 2  
    print(xu)  
print(xu) # 77 é impresso  
fun() # 2 é impresso  
print(xu) # 77 é impresso  
def alva():  
    global xu  
    xu=2  
alva()  
print(xu) # 2 é impresso
```

As boas práticas de programação sugerem não usar variáveis globais. Esta prática viola o encapsulamento e é fonte frequente de bugs de programação.

Seus módulos podem ser importados Organizando seu trabalho direitinho, você pode modularizar seus programas e importá-los quando necessário, acompanhe

```
#----- beta.py -----  
def alfa(x):  
    return x**3  
#-----  
>>> import beta  
>>> print(beta.alfa(4))  
#-----  
>>> from beta import *  
>>> print(alfa(4))
```

☞ Para você fazer

A seguir uma lista de funções que você deve programar em Python. Imprima as funções e devolva-as impressas junto com esta folha. Não esqueça de identificar com seu nome e o código 13 limite entrega 10/10/24

1. Escreva uma função que receba uma lista de números e devolva o maior deles. Se a entrada for 33, 22, -10, -4 e 33 a saída deve ser 33.
2. Escreva uma função que receba dois números e devolva `True` se tais números puderem indicar uma hora e minuto e `False` senão. Por exemplo, se a entrada for 23,59 a saída

deverá ser `True`, idem se for 0,0. Se a entrada for 33,5 a saída deverá ser `False`.

3. Escreva uma função que receba um valor numérico indicando o lado de um quadrado, medido em metros e devolva o valor do perímetro e da área do quadrado. Se a entrada for 1, a saída deve ser 4 e 1.
4. Escreva uma função que receba o valor de x e calcule e devolva o valor de $S = 1/2 + 2/3 + 3/4 + \dots + x/x + 1$. Se a entrada for 5 a saída deverá ser 3.55.
5. Escreva uma função que receba uma lista de números e imprima-a em ordem reversa (de trás para a frente). Se a entrada for 4, 7, -3, 22, 14, a impressão deverá ser de 14, 22, -3, 7 e 4.
6. Escreva uma função que receba dois números que descrevem o desempenho de alguém ao cursar a disciplina de MÉTODOS NUMÉRICOS. O primeiro número é a frequência ($0.0 \leq f \leq 100.0$) em porcentagem e o segundo número é a nota final obtida ($0.0 \leq NF \leq 10.0$). Deve imprimir a mensagem **APROVADO** ou **REPROVADO** a depender das regras da UFPR para isso. Por exemplo, se $f = 66$ e $NF = 7.1$ a mensagem deve ser **REPROVADO**.
7. Uma janela normanda tem o formato de um quadrado sobre o qual se coloca um semicírculo. Escreva uma função que receba o perímetro da janela e indique qual o lado do quadrado no qual ela está inserida. Por exemplo, se o perímetro da janela é de 10m, o lado do quadrado é 2,18m.
8. Escreva uma função que receba dois números: o preço do álcool e o preço da gasolina. Sabendo que um carro em particular faz um certo trajeto gastando 10l de álcool e 7l de gasolina (o mesmo trajeto), a função deve recomendar o combustível mais barato, a saber **ALCOOL** ou **GASOLINA**. Por exemplo, se $A=2.90$ R\$ e $G=4.10$ R\$, fazendo as contas, a gasolina é mais eficiente e a função deve recomendá-la.
9. Escreva uma função que receba um string contendo uma frase e devolva a quantidade de vogais que foram usadas na frase. Se a entrada for "Ivo viu a uva" a saída deverá ser 7.
10. Escreva uma função que receba uma lista de números e devolva a diferença entre o maior e o menor deles. Se a entrada for 33, 22, -10, -4 e 33 a saída deve ser 43.



408-75891 - e /nt

Parâmetros opcionais Seja uma função para sublinhar algo. Poderia ser

```
def sublinha():  
    print("_"*30)
```

Entretanto, nesta função tanto o caracter sublinha (`_`) quanto o valor 30 são fixos. Para permitir alterar isso, pode-se fazer:

```
def sublinha2(n=30, caractere="_"):  
    print(caractere*n)
```

Veja agora como a coisa é chamada

```
>>> def sublinha(n=30, cara="_"):  
    print(cara*n)  
>>> sublinha()
```

```
>>> sublinha(n=10)
```

```
>>> sublinha(cara='$')  
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
```

Funções em Python Além das centenas de funções pré-existentes em Python (print, min, max, sum, len, int, float, input...) a grande vantagem do ambiente é que você pode definir as funções que precisar para o seu trabalho. Depois que uma função foi definida ela passa a se comportar como se fosse (e é) uma função primitiva do Python.

Para definir uma nova função, usa-se o comando `def`, veja:

```
def nome(parâmetros):  
    comandos
```

Após a palavra `def` deve vir o nome (único) pelo qual a função será conhecida. Não pode ser uma palavra reservada do Python. Logo depois do nome, é obrigatório um "abre parênteses". A seguir uma quantidade opcional de parâmetros, que são como se fossem nomes de variáveis. Depois desses parâmetros vem o fecha parênteses. Se não houver parâmetro, depois do nome da função vem abre e fecha parênteses. Termina o cabeçalho o caráter "dois pontos" (:). Ele inicia um bloco que limita a definição da função.

Diferentemente do que aconteceu até agora, as linhas que fazem parte da função, não são executadas agora. Elas ficam armazenadas e serão executadas apenas quando a função agora definida, for chamada (executada).

A razão para definir uma função pode ser:

- * Para repetir o mesmo processamento em diversos locais de um programa.
- * Para encapsular um processamento específico.
- * Para comerciar ou apenas trocar algoritmos.
- * Para modularizar um grande processamento, permitindo que o mesmo seja desenvolvido por diversas pessoas.
- * Para poder manusear a complexidade crescente.

A função pode retornar algum resultado para quem a chama. Por exemplo, a função `len(lista)` retorna um valor que corresponde ao comprimento da lista que é passada como parâmetro.

Veja um exemplo de tudo que se disse

```
def maior(a):  
    i=len(a)-1 #1  
    mai=-99999 #2  
    while i>= 0: #3  
        if a[i]>mai: #4  
            mai=a[i] #5  
            i=i-1 #6  
    return mai #7  
    pass #8  
    #9
```

```
print(maior([1,5,7,2,3,17,5])) #10
```

Na linha #1 define-se a função de nome `maior` e se diz que ela deverá ser chamada com um único parâmetro. Quem fez a função sabe que este parâmetro único deve ser uma lista formada por números. Na linha #2 cria-se a variável `i` e atribui-se a ela o valor do tamanho da lista parâmetro menos 1. Como se sabe tal índice acessa o último elemento da lista. Na linha #3 define-se uma outra variável de controle `apt` a guardar o maior valor da lista até aquele momento. Como ainda nenhum item foi processado ela vale $-\infty$. Na linha #4 define-se que `i` vai variar até ter um valor negativo. Na linha #5 se o elemento de ordem `i` na lista for maior do que a variável `mai`, `mai` será atualizada com este novo valor. Na linha #7 que já está fora da indentação do `if`, mas ainda dentro da indentação do `while` o índice `i` é recuado uma posição e graças ao `while` há um retorno à linha #4. Finalmente, quando o ciclo `while` acabar, vai-se para a linha #8 que simplesmente devolve o valor calculado a quem chamou a função `maior` e o processamento desta função está encerrado. A linha #9 não é executada (de qualquer maneira `pass` não faz nada mesmo), mas lembre-se QUALQUER coisa colocada após um `return` é meramente decorativo, pois nunca vai ser executado. Na linha #10, já fora da função `maior` (note a indentação) a mesma vai ser executada. Veja que preciso colocar um `print` para externalizar o resultado da função. Se não houver `print`, a função vai ser executada, mas nada será visto de fora.

O parâmetro é uma lista ? Se o parâmetro for uma lista de 4 números, ele será um único parâmetro. Estruturalmente é diferente de 4 parâmetros escalares, que no fundo são uma lista de 4 números. Veja

```
def f1(L4):  
    print(L4[0], L4[1], L4[2], L4[3])  
# ----- OU -----  
def f2(a,b,c,d):  
    print(a, b, c, d)  
f1([1,2,3,4])  
f2(1,2,3,4)
```

Qual estratégia usar ao modularizar um problema e definir quantas e quais funções devem ser construídas é uma arte: a bússola nesta tarefa é a melhor maneira de reaproveitar código e de evitar a reinvenção da roada.

Variáveis locais e globais Como regra geral todas as variáveis definidas dentro de uma função (como `i` e `maior` no exemplo aí acima), são chamadas **locais** à função e isto significa que elas nunca sobrevivem ao tempo de execução da função: não existem antes da função e deixam de existir quando a função acaba. Desta maneira não podem ter seu valor consultado e/ou alterado por alguém externo à função.

Este fato permite que duas funções distintas tenham variáveis com o mesmo nome (como é muito comum um índice chamado `i`) sem que um atrapalhe o outro. Desta maneira a única forma de ceder e receber valores entre funções é através de parâmetros e de retornos.

Já uma variável que precise ser vista e alterada por outros é conhecida como variável **GLOBAL**. Ela deve ser definida dentro de um módulo, mas FORA de uma função. Agora ela pode ser manuseada por todas as funções desse módulo e por todos os módulos que importam o módulo que a definiu.

Para que uma função possa alterar uma variável global, a mesma deve ser definida como global usando-se o comando `GLOBAL` como linha seguinte à definição.

Veja um exemplo:

```
xu = 77  
def fun():  
    xu = 2  
    print(xu)  
print(xu) # 77 é impresso  
fun() # 2 é impresso  
print(xu) # 77 é impresso  
def alfa():  
    global xu  
    xu=2  
    alfa()  
print(xu) # 2 é impresso
```

As boas práticas de programação sugerem não usar variáveis globais. Esta prática viola o encapsulamento e é fonte frequente de bugs de programação.

Seus módulos podem ser importados Organizando seu trabalho direitinho, você pode modularizar seus programas e importá-los quando necessário, acompanhe

```
#----- beta.py -----  
def alfa(x):  
    return x**3  
#-----  
>>> import beta  
>>> print(beta.alfa(4))  
#-----  
>>> from beta import *  
>>> print(alfa(4))
```

☞ Para você fazer

A seguir uma lista de funções que você deve programar em Python. Imprima as funções e devolva-as impressas junto com esta folha. Não esqueça de identificar com seu nome e o código 14 limite entrega 10/10/24

1. Escreva uma função que receba uma lista de números e retorne a sua média. Por exemplo, se a entrada for 1, 2, 3, 4 a saída deverá ser 2.5.
2. Escreva uma função que receba dois números: o preço do álcool e o preço da gasolina. Sabendo que um carro em particular faz um certo trajeto gastando 10l de álcool e 7l de

gasolina (o mesmo trajeto), a função deve recomendar o combustível mais barato, a saber `ALCOOL` ou `GASOLINA`. Por exemplo, se `A=2.90 R$` e `G=4.10 R$`, fazendo as contas, a gasolina é mais eficiente e a função deve recomendá-la.

3. Uma janela normanda tem o formato de um quadrado sobre o qual se coloca um semicírculo. Escreva uma função que receba o perímetro da janela e indique qual o lado do quadrado no qual ela está inserida. Por exemplo, se o perímetro da janela é de 10m, o lado do quadrado é 2,18m.
4. Escreva uma função que receba uma lista de números e devolva a quantidade de números que são menores do que seus vizinhos. Os extremos da lista não devem ser testados pois lhes falta algum vizinho. Se a entrada for 1,4,2,8,22,50,100 a saída deverá ser 1.
5. Escreva uma função que receba uma lista de números positivos e negativos misturados e devolva duas listas: a primeira contendo apenas os positivos e a segunda apenas os negativos, na mesma ordem em que eles apareceram na entrada. Se a entrada for -2, 4, -4, -6, -11, 4, 18 as saídas deverão ser -2, -4, -6, -11 e 4, 4, 18.
6. Uma empresa de eletricidade cobra de seus clientes uma taxa base de R\$ 10 mensais mais seis centavos por quilowatt-hora para os primeiros 1200 quilowatt-hora e 7 centavos por quilowatt-hora para todo uso acima de 1.200. Escreva uma função que receba o consumo e indique o valor a ser cobrado. Por exemplo, para um consumo de 1000 Kwh, deve cobrar $10+60=70$ R\$.
7. Escreva uma função que receba dois números e devolva `True` se tais números puderem indicar uma hora e minuto e `False` senão. Por exemplo, se a entrada for 23,59 a saída deverá ser `True`, idem se for 0,0. Se a entrada for 33,5 a saída deverá ser `False`.
8. Escreva uma função que receba o valor de x e calcule e devolva o valor de $S = 1/4 + 2/9 + 3/16 + \dots + x/(x+1)^2$. Se a entrada for 5 a saída deverá ser 0.958611111.
9. Escreva uma função que receba um string contendo uma frase e devolva a quantidade de palavras que formam a frase. Se a entrada for "IVO VIU A UVA" a saída deverá ser 4.
10. Escreva uma função que receba a altura e o peso de uma pessoa. Deve calcular o `IMC` - índice de massa corporal ($IMV = peso/altura^2$) medidos em kg e m. A seguir deve emitir mensagem a depender do `IMC`. `IMC < 25=adequado`; `IMC < 30=sobrepeso`; `IMC < 35=obesidade` e `IMC ≥ 35=obesidade severa`. Por exemplo, se o sujeito pesa 110Kg e tem 1.70m de altura seu `IMC` é de 38.06. Logo o diagnóstico é **OBESIDADE SEVERA**.



408-75903 - e /nt

Parâmetros opcionais Seja uma função para sublinhar algo. Poderia ser

```
def sublinha():  
    print("_"*30)  
Entretanto, nesta função tanto o caracter sublinha (_) quanto o valor 30 são fixos. Para permitir alterar isso, pode-se fazer:  
def sublinha2(n=30, caractere="_"):  
    print(caractere*n)
```

Veja agora como a coisa é chamada

```
>>> def sublinha(n=30, cara="_"):  
    print(cara*n)  
>>> sublinha()
```

```
-----  
>>> sublinha(n=10)  
-----  
>>> sublinha(cara='$')  
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
```

Funções em Python Além das centenas de funções pré-existentes em Python (print, min, max, sum, len, int, float, input...) a grande vantagem do ambiente é que você pode definir as funções que precisar para o seu trabalho. Depois que uma função foi definida ela passa a se comportar como se fosse (e é) uma função primitiva do Python.

Para definir uma nova função, usa-se o comando `def`, veja:

```
def nome(parâmetros):  
    comandos
```

Após a palavra `def` deve vir o nome (único) pelo qual a função será conhecida. Não pode ser uma palavra reservada do Python. Logo depois do nome, é obrigatório um "abre parênteses". A seguir uma quantidade opcional de parâmetros, que são como se fossem nomes de variáveis. Depois desses parâmetros vem o fecha parênteses. Se não houver parâmetro, depois do nome da função vem abre e fecha parênteses. Termina o cabeçalho o caráter "dois pontos" (:). Ele inicia um bloco que limita a definição da função.

Diferentemente do que aconteceu até agora, as linhas que fazem parte da função, não são executadas agora. Elas ficam armazenadas e serão executadas apenas quando a função agora definida, for chamada (executada).

A razão para definir uma função pode ser:

- * Para repetir o mesmo processamento em diversos locais de um programa.
- * Para encapsular um processamento específico.
- * Para comercializar ou apenas trocar algoritmos.
- * Para modularizar um grande processamento, permitindo que o mesmo seja desenvolvido por diversas pessoas.
- * Para poder manusear a complexidade crescente.

A função pode retornar algum resultado para quem a chama. Por exemplo, a função `len(lista)` retorna um valor que corresponde ao comprimento da lista que é passada como parâmetro.

Veja um exemplo de tudo que se disse

```
def maior(a):  
    #1  
    i=len(a)-1 #2  
    mai=-99999 #3  
    while i>= 0:  
        if a[i]>mai: #5  
            mai=a[i] #6  
            i=i-1 #7  
    return mai #8  
    pass #9  
print(maior([1,5,7,2,3,17,5]))
```

Na linha #1 define-se a função de nome `maior` e se diz que ela deverá ser chamada com um único parâmetro. Quem fez a função sabe que este parâmetro único deve ser uma lista formada por números. Na linha #2 cria-se a variável `i` e atribui-se a ela o valor do tamanho da lista parâmetro menos 1. Como se sabe tal índice acessa o último elemento da lista. Na linha #3 define-se uma outra variável de controle apta a guardar o maior valor da lista até aquele momento. Como ainda nenhum item foi processado ela vale $-\infty$. Na linha #4 define-se que `i` vai variar até ter um valor negativo. Na linha #5 se o elemento de ordem `i` na lista for maior do que a variável `mai`, `mai` será atualizada com este novo valor. Na linha #7 que já está fora da indentação do `if`, mas ainda dentro da indentação do `while` o índice `i` é recuado uma posição e graças ao `while` há um retorno à linha #4. Finalmente, quando o ciclo `while` acabar, vai-se para a linha #8 que simplesmente devolve o valor calculado a quem chamou a função `maior` e o processamento desta função está encerrado. A linha #9 não é executada (de qualquer maneira `pass` não faz nada mesmo), mas lembre-se QUALQUER coisa colocada após um `return` é meramente decorativo, pois nunca vai ser executado. Na linha #10, já fora da função `maior` (note a indentação) a mesma vai ser executada. Veja que preciso colocar um `print` para externalizar o resultado da função. Se não houver `print`, a função vai ser executada, mas nada será visto de fora.

O parâmetro é uma lista ? Se o parâmetro for uma lista de 4 números, ele será um único parâmetro. Estruturalmente é diferente de 4 parâmetros escalares, que no fundo são uma lista de 4 números. Veja

```
def f1(L4):  
    print(L4[0], L4[1], L4[2], L4[3])  
# ----- OU -----  
def f2(a,b,c,d):  
    print(a, b, c, d)  
f1([1,2,3,4])  
f2(1,2,3,4)
```

Qual estratégia usar ao modularizar um problema e definir quantas e quais funções devem ser construídas é uma arte: a bússola nesta tarefa é a melhor maneira de reaproveitar código e de evitar a reinvenção da roda.

Variáveis locais e globais Como regra geral todas as variáveis definidas dentro de uma função (como `i` e maior no exemplo aí acima), são chamadas **locais** à função e isto significa que elas nunca sobrevivem ao tempo de execução da função: não existem antes da função e deixam de existir quando a função acaba. Desta maneira não podem ter seu valor consultado e/ou alterado por alguém externo à função.

Este fato permite que duas funções distintas tenham variáveis com o mesmo nome (como é muito comum um índice chamado `i`) sem que um atrapalhe o outro. Desta maneira a única forma de ceder e receber valores entre funções é através de parâmetros e de retornos.

Já uma variável que precise ser vista e alterada por outros é conhecida como variável **GLOBAL**. Ela deve ser definida dentro de um módulo, mas **FORA** de uma função. Agora ela pode ser manuseada por todas as funções desse módulo e por todos os módulos que importam o módulo que a definiu.

Para que uma função possa alterar uma variável global, a mesma deve ser definida como global usando-se o comando `GLOBAL` como linha seguinte à definição.

Veja um exemplo:

```
xu = 77  
def fun():  
    xu = 2  
    print(xu)  
print(xu) # 77 é impresso  
fun() # 2 é impresso  
print(xu) # 77 é impresso  
def alva():  
    global xu  
    xu=2  
alva()  
print(xu) # 2 é impresso
```

As boas práticas de programação sugerem não usar variáveis globais. Esta prática viola o encapsulamento e é fonte frequente de bugs de programação.

Seus módulos podem ser importados

Organizando seu trabalho direitinho, você pode modularizar seus programas e importá-los quando necessário, acompanhe

```
#----- beta.py -----  
def alfa(x):  
    return x**3  
#-----  
>>> import beta  
>>> print(beta.alfa(4))  
#-----  
>>> from beta import *  
>>> print(alfa(4))
```

👉 Para você fazer

A seguir uma lista de funções que você deve programar em Python. Imprima as funções e devolva-as impressas junto com esta folha. Não esqueça de identificar com seu nome e o código 15 limite entrega 10/10/24

1. Uma empresa de eletricidade cobra de seus clientes uma taxa base de R\$ 10 mensais mais seis centavos por quilowatt-hora para os primeiros 1200 quilowatt-hora e 7 centavos por quilowatt-hora para todo uso acima de 1.200. Escreva uma função que receba o consumo e indique o valor a ser cobrado.

Por exemplo, para um consumo de 1000 Kwh, deve cobrar $10+60=70$ R\$.

2. Uma janela normanda tem o formato de um quadrado sobre o qual se coloca um semicírculo. Escreva uma função que receba o perímetro da janela e indique qual o lado do quadrado no qual ela está inserida. Por exemplo, se o perímetro da janela é de 10m, o lado do quadrado é 2,18m.
3. Escreva uma função que receba 2 horários (no formato HH, MM) indicando o início e o fim de uma determinada tarefa. A função deve devolver a duração em minutos da tarefa. Se a entrada for 1,10 e 2,0 a saída deverá ser 50.
4. Escreva uma função que receba um valor numérico indicando o lado de um quadrado, medido em metros e devolva o valor do perímetro e da área do quadrado. Se a entrada for 1, a saída deve ser 4 e 1.
5. Escreva uma função que receba 2 valores: o peso e a altura de alguém e devolva o IMC (Índice de Massa Corporal): $IMC = peso \div (altura)^2$. Peso em Kg e altura em m. Se a entrada for 80, 1.80 a saída deverá ser 24.69.
6. Escreva uma função que receba dois números que descrevem o desempenho de alguém ao cursar a disciplina de MÉTODOS NUMÉRICOS. O primeiro número é a frequência ($0.0 \leq f \leq 100.0$) em porcentagem e o segundo número é a nota final obtida ($0.0 \leq NF \leq 10.0$). Deve imprimir a mensagem **APROVADO** ou **REPROVADO** a depender das regras da UFPR para isso. Por exemplo, se $f = 66$ e $NF = 7.1$ a mensagem deve ser **REPROVADO**.
7. Escreva uma função que receba uma lista com 3 valores numéricos: A, B e C e calcule as raízes de $Ax^2 + Bx + c = 0$, usando a fórmula de Baskara. Se a entrada for 1, -5 e 6 a resposta deverá ser 2 e 3.
8. Escreva uma função que receba dois números e devolva `True` se tais números puderem indicar um dia e mês e `False` senão. Por exemplo, se a entrada for 23,11 a saída deverá ser `True`, idem se for 1,1. Se a entrada for 33,5 a saída deverá ser `False`.
9. Escreva uma função que receba uma lista de números e devolva `True` se a lista estiver em ordem crescente e `False` senão. Uma lista X_i está em ordem crescente se $x_{i+1} \geq x_i$. Se a entrada for 1,2,2,2,2,2,3 a saída deverá ser `True`.
10. Escreva uma função que receba um número inteiro x e calcule e devolva a soma dos múltiplos de 13 menores que x . Se a entrada for 100, a saída deverá ser 364 que é igual à soma de 13 26 39 52 65 78 91.



408-75910 - e /nt

Parâmetros opcionais Seja uma função para sublinhar algo. Poderia ser

```
def sublinha():  
    print("_"*30)  
Entretanto, nesta função tanto o caracter sublinha (_) quanto o valor 30 são fixos. Para permitir alterar isso, pode-se fazer:  
def sublinha2(n=30, caractere="_"):  
    print(caractere*n)
```

Veja agora como a coisa é chamada

```
>>> def sublinha(n=30, cara="_"):  
    print(cara*n)  
>>> sublinha()
```

```
-----  
>>> sublinha(n=10)  
-----  
>>> sublinha(cara='$')  
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
```

Funções em Python Além das centenas de funções pré-existentes em Python (print, min, max, sum, len, int, float, input...) a grande vantagem do ambiente é que você pode definir as funções que precisar para o seu trabalho. Depois que uma função foi definida ela passa a se comportar como se fosse (e é) uma função primitiva do Python.

Para definir uma nova função, usa-se o comando `def`, veja:

```
def nome(parâmetros):  
    comandos
```

Após a palavra `def` deve vir o nome (único) pelo qual a função será conhecida. Não pode ser uma palavra reservada do Python. Logo depois do nome, é obrigatório um “abre parênteses”. A seguir uma quantidade opcional de parâmetros, que são como se fossem nomes de variáveis. Depois desses parâmetros vem o fecha parênteses. Se não houver parâmetro, depois do nome da função vem abre e fecha parênteses. Termina o cabeçalho o caráter “dois pontos” (:). Ele inicia um bloco que limita a definição da função.

Diferentemente do que aconteceu até agora, as linhas que fazem parte da função, não são executadas agora. Elas ficam armazenadas e serão executadas apenas quando a função agora definida, for chamada (executada).

A razão para definir uma função pode ser:

- * Para repetir o mesmo processamento em diversos locais de um programa.
- * Para encapsular um processamento específico.
- * Para comerciar ou apenas trocar algoritmos.
- * Para modularizar um grande processamento, permitindo que o mesmo seja desenvolvido por diversas pessoas.
- * Para poder manusear a complexidade crescente.

A função pode retornar algum resultado para quem a chama. Por exemplo, a função `len(lista)` retorna um valor que corresponde ao comprimento da lista que é passada como parâmetro.

Veja um exemplo de tudo que se disse

```
def maior(a):  
    i=len(a)-1 #1  
    mai=-99999 #2  
    while i>= 0: #3  
        if a[i]>mai: #4  
            mai=a[i] #5  
            i=i-1 #6  
    return mai #7  
    pass #8  
print(maior([1,5,7,2,3,17,5])) #9
```

Na linha #1 define-se a função de nome `maior` e se diz que ela deverá ser chamada com um único parâmetro. Quem fez a função sabe que este parâmetro único deve ser uma lista formada por números. Na linha #2 cria-se a variável `i` e atribui-se a ela o valor do tamanho da lista parâmetro menos 1. Como se sabe tal índice acessa o último elemento da lista. Na linha #3 define-se uma outra variável de controle apta a guardar o maior valor da lista até aquele momento. Como ainda nenhum item foi processado ela vale $-\infty$. Na linha #4 define-se que `i` vai variar até ter um valor negativo. Na linha #5 se o elemento de ordem `i` na lista for maior do que a variável `mai`, `mai` será atualizada com este novo valor. Na linha #7 que já está fora da indentação do `if`, mas ainda dentro da indentação do `while` o índice `i` é recuado uma posição e graças ao `while` há um retorno à linha #4. Finalmente, quando o ciclo `while` acabar, vai-se para a linha #8 que simplesmente devolve o valor calculado a quem chamou a função `maior` e o processamento desta função está encerrado. A linha #9 não é executada (de qualquer maneira `pass` não faz nada mesmo), mas lembre-se QUALQUER coisa colocada após um `return` é meramente decorativo, pois nunca vai ser executado. Na linha #10, já fora da função `maior` (note a indentação) a mesma vai ser executada. Veja que preciso colocar um `print` para externalizar o resultado da função. Se não houver `print`, a função vai ser executada, mas nada será visto de fora.

O parâmetro é uma lista ? Se o parâmetro for uma lista de 4 números, ele será um único parâmetro. Estruturalmente é diferente de 4 parâmetros escalares, que no fundo são uma lista de 4 números. Veja

```
def f1(L4):  
    print(L4[0], L4[1], L4[2], L4[3])  
# ---- OU ----  
def f2(a,b,c,d):  
    print(a, b, c, d)  
f1([1,2,3,4])  
f2(1,2,3,4)
```

Qual estratégia usar ao modularizar um problema e definir quantas e quais funções devem ser construídas é uma arte: a bússola nesta tarefa é a melhor maneira de reaproveitar código e de evitar a reinvenção da roda.

Variáveis locais e globais Como regra geral todas as variáveis definidas dentro de uma função (como `i` e `maior` no exemplo aí acima), são chamadas **locais** à função e isto significa que elas nunca sobrevivem ao tempo de execução da função: não existem antes da função e deixam de existir quando a função acaba. Desta maneira não podem ter seu valor consultado e/ou alterado por alguém externo à função.

Este fato permite que duas funções distintas tenham variáveis com o mesmo nome (como é muito comum um índice chamado `i`) sem que um atrapalhe o outro. Desta maneira a única forma de ceder e receber valores entre funções é através de parâmetros e de retornos.

Já uma variável que precise ser vista e alterada por outros é conhecida como variável **GLOBAL**. Ela deve ser definida dentro de um módulo, mas **FORA** de uma função. Agora ela pode ser manuseada por todas as funções desse módulo e por todos os módulos que importam o módulo que a definiu.

Para que uma função possa alterar uma variável global, a mesma deve ser definida como global usando-se o comando `GLOBAL` como linha seguinte à definição.

Veja um exemplo:

```
xu = 77  
def fun():  
    xu = 2  
    print(xu)  
print(xu) # 77 é impresso  
fun() # 2 é impresso  
print(xu) # 77 é impresso  
def alva():  
    global xu  
    xu=2  
alva()  
print(xu) # 2 é impresso
```

As boas práticas de programação sugerem não usar variáveis globais. Esta prática viola o encapsulamento e é fonte frequente de bugs de programação.

Seus módulos podem ser importados Organizando seu trabalho direitinho, você pode modularizar seus programas e importá-los quando necessário, acompanhe

```
#----- beta.py -----  
def alfa(x):  
    return x**3  
#-----  
>>> import beta  
>>> print(beta.alfa(4))  
#-----  
>>> from beta import *  
>>> print(alfa(4))
```

📖 Para você fazer

A seguir uma lista de funções que você deve programar em Python. Imprima as funções e devolva-as impressas junto com esta folha. Não esqueça de identificar com seu nome e o código 16 limite entrega 10/10/24

1. Escreva uma função que receba uma lista de números e devolva a diferença entre o maior e o menor deles. Se a entrada for 33, 22, -10, -4 e 33 a saída deve ser 43.
2. Escreva uma função que receba um valor numérico indicando o lado de um quadrado,

medido em metros e devolva o valor do perímetro e da área do quadrado. Se a entrada for 1, a saída deve ser 4 e 1.

3. Um plano de telefonia celular tem uma taxa de R\$ 35 por mês. O plano inclui 400 minutos gratuitos e taxa de 10 centavos para cada minuto adicional utilizado. Escreva uma função que indique o custo mensal em função do número de minutos utilizados. Por exemplo se a função for chamada com 410, deve responder 36 R\$.
4. Escreva uma função que receba uma lista de números e devolva o maior deles. Se a entrada for 33, 22, -10, -4 e 33 a saída deve ser 33.
5. Escreva uma função que receba uma lista de números e devolva o menor deles. Se a entrada for 33, 22, -10, -4 e 33 a saída deve ser -10.
6. Escreva uma função que receba uma lista de números e devolva a quantidade de números que são menores do que seus vizinhos. Os extremos da lista não devem ser testados pois lhes falta algum vizinho. Se a entrada for 1,4,2,8,22,50,100 a saída deverá ser 1.
7. Escreva uma função que receba um string contendo uma frase e devolva a quantidade de palavras que formam a frase. Se a entrada for "IVO VIU A UVA" a saída deverá ser 4.
8. Uma empresa de eletricidade cobra de seus clientes uma taxa base de R\$ 10 mensais mais seis centavos por quilowatt-hora para os primeiros 1200 quilowatt-hora e 7 centavos por quilowatt-hora para todo uso acima de 1.200. Escreva uma função que receba o consumo e indique o valor a ser cobrado. Por exemplo, para um consumo de 1000 Kwh, deve cobrar $10+60=70$ R\$.
9. Escreva uma função que receba um parâmetro numérico e retorne a palavra `PAR` ou a palavra `IMPAR` se o parâmetro for par ou ímpar. Por exemplo, se a entrada for 3, a saída deverá ser `IMPAR`.
10. Uma janela normanda tem o formato de um quadrado sobre o qual se coloca um semicírculo. Escreva uma função que receba o perímetro da janela e indique qual o lado do quadrado no qual ela está inserida. Por exemplo, se o perímetro da janela é de 10m, o lado do quadrado é 2,18m.



408-75927 - e /nt

Parâmetros opcionais Seja uma função para sublinhar algo. Poderia ser

```
def sublinha():  
    print("_"*30)  
Entretanto, nesta função tanto o caracter sublinha (_) quanto o valor 30 são fixos. Para permitir alterar isso, pode-se fazer:  
def sublinha2(n=30, caractere="_"):  
    print(caractere*n)  
Veja agora como a coisa é chamada  
>>> def sublinha(n=30, cara="_"):  
        print(cara*n)  
>>> sublinha()  
-----  
>>> sublinha(n=10)  
-----  
>>> sublinha(cara='$!')  
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
```

Funções em Python Além das centenas de funções pré-existentes em Python (print, min, max, sum, len, int, float, input...) a grande vantagem do ambiente é que você pode definir as funções que precisar para o seu trabalho. Depois que uma função foi definida ela passa a se comportar como se fosse (e é) uma função primitiva do Python.

Para definir uma nova função, usa-se o comando `def`, veja:

```
def nome(parâmetros):  
    comandos
```

Após a palavra `def` deve vir o nome (único) pelo qual a função será conhecida. Não pode ser uma palavra reservada do Python. Logo depois do nome, é obrigatório um “abre parênteses”. A seguir uma quantidade opcional de parâmetros, que são como se fossem nomes de variáveis. Depois desses parâmetros vem o fecha parênteses. Se não houver parâmetro, depois do nome da função vem abre e fecha parênteses. Termina o cabeçalho o caráter “dois pontos” (:). Ele inicia um bloco que limita a definição da função.

Diferentemente do que aconteceu até agora, as linhas que fazem parte da função, não são executadas agora. Elas ficam armazenadas e serão executadas apenas quando a função agora definida, for chamada (executada).

A razão para definir uma função pode ser:

- * Para repetir o mesmo processamento em diversos locais de um programa.
- * Para encapsular um processamento específico.
- * Para comerciar ou apenas trocar algoritmos.
- * Para modularizar um grande processamento, permitindo que o mesmo seja desenvolvido por diversas pessoas.
- * Para poder manusear a complexidade crescente.

A função pode retornar algum resultado para quem a chama. Por exemplo, a função `len(lista)` retorna um valor que corresponde ao comprimento da lista que é passada como parâmetro.

Veja um exemplo de tudo que se disse

```
def maior(a):  
    i=len(a)-1 #1  
    mai=-99999 #2  
    while i>= 0: #3  
        if a[i]>mai: #4  
            mai=a[i] #5  
            i=i-1 #6  
    return mai #7  
    pass #8  
print(maior([1,5,7,2,3,17,5])) #9
```

Na linha #1 define-se a função de nome `maior` e se diz que ela deverá ser chamada com um único parâmetro. Quem fez a função sabe que este parâmetro único deve ser uma lista formada por números. Na linha #2 cria-se a variável `i` e atribui-se a ela o valor do tamanho da lista parâmetro menos 1. Como se sabe tal índice acessa o último elemento da lista. Na linha #3 define-se uma outra variável de controle apta a guardar o maior valor da lista até aquele momento. Como ainda nenhum item foi processado ela vale $-\infty$. Na linha #4 define-se que `i` vai variar até ter um valor negativo. Na linha #5 se o elemento de ordem `i` na lista for maior do que a variável `mai`, `mai` será atualizada com este novo valor. Na linha #7 que já está fora da indentação do `if`, mas ainda dentro da indentação do `while` o índice `i` é recuado uma posição e graças ao `while` há um retorno à linha #4. Finalmente, quando o ciclo `while` acabar, vai-se para a linha #8 que simplesmente devolve o valor calculado a quem chamou a função `maior` e o processamento desta função está encerrado. A linha #9 não é executada (de qualquer maneira `pass` não faz nada mesmo), mas lembre-se QUALQUER coisa colocada após um `return` é meramente decorativo, pois nunca vai ser executado. Na linha #10, já fora da função `maior` (note a indentação) a mesma vai ser executada. Veja que preciso colocar um `print` para externalizar o resultado da função. Se não houver `print`, a função vai ser executada, mas nada será visto de fora.

O parâmetro é uma lista ? Se o parâmetro for uma lista de 4 números, ele será um único parâmetro. Estruturalmente é diferente de 4 parâmetros escalares, que no fundo são uma lista de 4 números. Veja

```
def f1(L4):  
    print(L4[0], L4[1], L4[2], L4[3])  
# ----- OU -----  
def f2(a,b,c,d):  
    print(a, b, c, d)  
f1([1,2,3,4])  
f2(1,2,3,4)
```

Qual estratégia usar ao modularizar um problema e definir quantas e quais funções devem ser construídas é uma arte: a bússola nesta tarefa é a melhor maneira de reaproveitar código e de evitar a reinvenção da roada.

Variáveis locais e globais Como regra geral todas as variáveis definidas dentro de uma função (como `i` e `maior` no exemplo aí acima), são chamadas **locais** à função e isto significa que elas nunca sobrevivem ao tempo de execução da função: não existem antes da função e deixam de existir quando a função acaba. Desta maneira não podem ter seu valor consultado e/ou alterado por alguém externo à função.

Este fato permite que duas funções distintas tenham variáveis com o mesmo nome (como é muito comum um índice chamado `i`) sem que um atrapalhe o outro. Desta maneira a única forma de ceder e receber valores entre funções é através de parâmetros e de retornos.

Já uma variável que precise ser vista e alterada por outros é conhecida como variável **GLOBAL**. Ela deve ser definida dentro de um módulo, mas **FORA** de uma função. Agora ela pode ser manuseada por todas as funções desse módulo e por todos os módulos que importam o módulo que a definiu.

Para que uma função possa alterar uma variável global, a mesma deve ser definida como global usando-se o comando `GLOBAL` como linha seguinte à definição.

Veja um exemplo:

```
xu = 77  
def fun():  
    xu = 2  
    print(xu)  
print(xu) # 77 é impresso  
fun() # 2 é impresso  
print(xu) # 77 é impresso  
def alva():  
    global xu  
    xu=2  
alva()  
print(xu) # 2 é impresso
```

As boas práticas de programação sugerem não usar variáveis globais. Esta prática viola o encapsulamento e é fonte frequente de bugs de programação.

Seus módulos podem ser importados Organizando seu trabalho direitinho, você pode modularizar seus programas e importá-los quando necessário, acompanhe

```
#----- beta.py -----  
def alfa(x):  
    return x**3  
#-----  
>>> import beta  
>>> print(beta.alfa(4))  
#-----  
>>> from beta import *  
>>> print(alfa(4))
```

☞ Para você fazer

A seguir uma lista de funções que você deve programar em Python. Imprima as funções e devolva-as impressas junto com esta folha. Não esqueça de identificar com seu nome e o código 17 limite entrega 10/10/24

1. Escreva uma função que receba um número inteiro `x` e calcule e devolva a soma dos múltiplos de 13 menores que `x`. Se a entrada for 100, a saída deverá ser 364 que é igual à soma de 13 26 39 52 65 78 91.
2. Escreva uma função que receba um parâmetro numérico e retorne a palavra `PAR` ou a palavra `IMPAR` se o parâmetro for par ou

impar. Por exemplo, se a entrada for 3, a saída deverá ser `IMPAR`.

3. Escreva uma função que receba um valor numérico indicando o lado de um quadrado, medido em metros e devolva o valor do perímetro e da área do quadrado. Se a entrada for 1, a saída deve ser 4 e 1.
4. Escreva uma função que receba uma lista de números e imprima-a em ordem reversa (de trás para a frente). Se a entrada for 4, 7, -3, 22, 14, a impressão deverá ser de 14, 22, -3, 7 e 4.
5. Escreva uma função que receba dois números que descrevem o desempenho de alguém ao cursar a disciplina de **MÉTODOS NUMÉRICOS**. O primeiro número é a frequência ($0.0 \leq f \leq 100.0$) em percentagem e o segundo número é a nota final obtida ($0.0 \leq NF \leq 10.0$). Deve imprimir a mensagem `APROVADO` ou `REPROVADO` a depender das regras da UFPR para isso. Por exemplo, se $f = 66$ e $NF = 7.1$ a mensagem deve ser `REPROVADO`.
6. Escreva uma função que receba dois números e devolva `True` se tais números puderem indicar um dia e mês e `False` senão. Por exemplo, se a entrada for 23,11 a saída deverá ser `True`, idem se for 1,1. Se a entrada for 33,5 a saída deverá ser `False`.
7. Escreva uma função que receba um número inteiro `x` e devolva `True` se ele for primo e `False` senão. Um número é primo quando seus únicos divisores são ele mesmo e a unidade. Se a entrada for 17 a saída deverá ser `True`.
8. Escreva uma função que receba dois números: o preço do álcool e o preço da gasolina. Sabendo que um carro em particular faz um certo trajeto gastando 10l de álcool e 7l de gasolina (o mesmo trajeto), a função deve recomendar o combustível mais barato, a saber `ALCOOL` ou `GASOLINA`. Por exemplo, se $A=2.90$ R\$ e $G=4.10$ R\$, fazendo as contas, a gasolina é mais eficiente e a função deve recomendá-la.
9. Escreva uma função que receba uma lista de números e devolva o menor deles. Se a entrada for 33, 22, -10, -4 e 33 a saída deve ser -10.
10. Escreva uma função que receba uma lista de números e retorne a sua média. Por exemplo, se a entrada for 1, 2, 3, 4 a saída deverá ser 2.5.



408-75934 - e /nt

Parâmetros opcionais Seja uma função para sublinhar algo. Poderia ser

```
def sublinha():  
    print("_"*30)
```

Entretanto, nesta função tanto o caracter sublinha (`_`) quanto o valor 30 são fixos. Para permitir alterar isso, pode-se fazer:

```
def sublinha2(n=30, caractere="_"):  
    print(caractere*n)
```

Veja agora como a coisa é chamada

```
>>> def sublinha(n=30, cara="_"):  
    print(cara*n)  
>>> sublinha()
```

```
-----  
>>> sublinha(n=10)
```

```
-----  
>>> sublinha(cara='$')  
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
```

Funções em Python Além das centenas de funções pré-existentes em Python (print, min, max, sum, len, int, float, input...) a grande vantagem do ambiente é que você pode definir as funções que precisar para o seu trabalho. Depois que uma função foi definida ela passa a se comportar como se fosse (e é) uma função primitiva do Python.

Para definir uma nova função, usa-se o comando `def`, veja:

```
def nome(parâmetros):  
    comandos
```

Após a palavra `def` deve vir o nome (único) pelo qual a função será conhecida. Não pode ser uma palavra reservada do Python. Logo depois do nome, é obrigatório um “abre parênteses”. A seguir uma quantidade opcional de parâmetros, que são como se fossem nomes de variáveis. Depois desses parâmetros vem o fecha parênteses. Se não houver parâmetro, depois do nome da função vem abre e fecha parênteses. Termina o cabeçalho o caráter “dois pontos” (:). Ele inicia um bloco que limita a definição da função.

Diferentemente do que aconteceu até agora, as linhas que fazem parte da função, não são executadas agora. Elas ficam armazenadas e serão executadas apenas quando a função agora definida, for chamada (executada).

A razão para definir uma função pode ser:

- * Para repetir o mesmo processamento em diversos locais de um programa.
- * Para encapsular um processamento específico.
- * Para comercializar ou apenas trocar algoritmos.
- * Para modularizar um grande processamento, permitindo que o mesmo seja desenvolvido por diversas pessoas.
- * Para poder manusear a complexidade crescente.

A função pode retornar algum resultado para quem a chama. Por exemplo, a função `len(lista)` retorna um valor que corresponde ao comprimento da lista que é passada como parâmetro.

Veja um exemplo de tudo que se disse

```
def maior(a):  
    #1          #1  
    i=len(a)-1  #2  
    mai=-99999  #3  
    while i>= 0:  
        #4  
        if a[i]>mai:  
            #5  
            mai=a[i] #6  
            i=i-1     #7  
    return mai     #8  
    pass          #9
```

```
print(maior([1,5,7,2,3,17,5]))
```

Na linha #1 define-se a função de nome `maior` e se diz que ela deverá ser chamada com um único parâmetro. Quem fez a função sabe que este parâmetro único deve ser uma lista formada por números. Na linha #2 cria-se a variável `i` e atribui-se a ela o valor do tamanho da lista parâmetro menos 1. Como se sabe tal índice acessa o último elemento da lista. Na linha #3 define-se uma outra variável de controle apta a guardar o maior valor da lista até aquele momento. Como ainda nenhum item foi processado ela vale $-\infty$. Na linha #4 define-se que `i` vai variar até ter um valor negativo. Na linha #5 se o elemento de ordem `i` na lista for maior do que a variável `mai`, `mai` será atualizada com este novo valor. Na linha #7 que já está fora da indentação do `if`, mas ainda dentro da indentação do `while` o índice `i` é recuado uma posição e graças ao `while` há um retorno à linha #4. Finalmente, quando o ciclo `while` acabar, vai-se para a linha #8 que simplesmente devolve o valor calculado a quem chamou a função `maior` e o processamento desta função está encerrado. A linha #9 não é executada (de qualquer maneira `pass` não faz nada mesmo), mas lembre-se QUALQUER coisa colocada após um `return` é meramente decorativo, pois nunca vai ser executado. Na linha #10, já fora da função `maior` (note a indentação) a mesma vai ser executada. Veja que preciso colocar um `print` para externalizar o resultado da função. Se não houver `print`, a função vai ser executada, mas nada será visto de fora.

O parâmetro é uma lista ? Se o parâmetro for uma lista de 4 números, ele será um único parâmetro. Estruturalmente é diferente de 4 parâmetros escalares, que no fundo são uma lista de 4 números. Veja

```
def f1(L4):  
    print(L4[0], L4[1], L4[2], L4[3])  
# ----- OU -----  
def f2(a,b,c,d):  
    print(a, b, c, d)  
f1([1,2,3,4])  
f2(1,2,3,4)
```

Qual estratégia usar ao modularizar um problema e definir quantas e quais funções devem ser construídas é uma arte: a bússola nesta tarefa é a melhor maneira de reaproveitar código e de evitar a reinvenção da roda.

Variáveis locais e globais Como regra geral todas as variáveis definidas dentro de uma função (como `i` e `maior` no exemplo aí acima), são chamadas **locais** à função e isto significa que elas nunca sobrevivem ao tempo de execução da função: não existem antes da função e deixam de existir quando a função acaba. Desta maneira não podem ter seu valor consultado e/ou alterado por alguém externo à função.

Este fato permite que duas funções distintas tenham variáveis com o mesmo nome (como é muito comum um índice chamado `i`) sem que um atrapalhe o outro. Desta maneira a única forma de ceder e receber valores entre funções é através de parâmetros e de retornos.

Já uma variável que precise ser vista e alterada por outros é conhecida como variável **GLOBAL**. Ela deve ser definida dentro de um módulo, mas **FORA** de uma função. Agora ela pode ser manuseada por todas as funções desse módulo e por todos os módulos que importam o módulo que a definiu.

Para que uma função possa alterar uma variável global, a mesma deve ser definida como global usando-se o comando **GLOBAL** como linha seguinte à definição.

Veja um exemplo:

```
xu = 77  
def fun():  
    xu = 2  
    print(xu)  
print(xu) # 77 é impresso  
fun()     # 2 é impresso  
print(xu) # 77 é impresso  
def alva():  
    global xu  
    xu=2  
alva()  
print(xu) # 2 é impresso
```

As boas práticas de programação sugerem não usar variáveis globais. Esta prática viola o encapsulamento e é fonte frequente de bugs de programação.

Seus módulos podem ser importados Organizando seu trabalho direitinho, você pode modularizar seus programas e importá-los quando necessário, acompanhe

```
#----- beta.py -----  
def alfa(x):  
    return x**3  
#-----  
>>> import beta  
>>> print(beta.alfa(4))  
#-----  
>>> from beta import *  
>>> print(alfa(4))
```

☞ Para você fazer

A seguir uma lista de funções que você deve programar em Python. Imprima as funções e devolva-as impressas junto com esta folha. Não esqueça de identificar com seu nome e o código 18 limite entrega 10/10/24

1. Escreva uma função que receba um string contendo uma frase e devolva a quantidade de vogais que foram usadas na frase. Se a entrada for "Ivo viu a uva" a saída deverá ser 7.
2. Escreva uma função que receba um número inteiro `x` e calcule e devolva a soma dos múltiplos de 13 menores que `x`. Se a entrada

for 100, a saída deverá ser 364 que é igual à soma de 13 26 39 52 65 78 91.

3. Escreva uma função que receba uma lista de números positivos e negativos misturados e devolva duas listas: a primeira contendo apenas os positivos e a segunda apenas os negativos, na mesma ordem em que eles apareceram na entrada. Se a entrada for -2, 4, -4, -6, -11, 4, 18 as saídas deverão ser -2, -4, -6, -11 e 4, 18.
4. Escreva uma função que receba um parâmetro numérico e retorne a palavra **PAR** ou a palavra **IMPAR** se o parâmetro for par ou ímpar. Por exemplo, se a entrada for 3, a saída deverá ser **IMPAR**.
5. Escreva uma função que receba um string contendo uma frase e devolva a quantidade de palavras masculinas que formam a frase. Por hipótese, as palavras masculinas são as que terminam em "O" ou "o". Se a entrada for "O RATO ROEU A ROUPA DO REI" a saída deverá ser 3.
6. Escreva uma função que receba uma lista de números e retorne a sua média. Por exemplo, se a entrada for 1, 2, 3, 4 a saída deverá ser 2.5.
7. Uma empresa de eletricidade cobra de seus clientes uma taxa base de R\$ 10 mensais mais seis centavos por quilowatt-hora para os primeiros 1200 quilowatt-hora e 7 centavos por quilowatt-hora para todo uso acima de 1.200. Escreva uma função que receba o consumo e indique o valor a ser cobrado. Por exemplo, para um consumo de 1000 Kwh, deve cobrar 10+60=70 R\$.
8. Escreva uma função que receba uma lista de números e devolva **True** se a lista estiver em ordem crescente e **False** senão. Uma lista X_i está em ordem crescente se $x_{i+1} \geq x_i$. Se a entrada for 1,2,2,2,2,2,3 a saída deverá ser **True**.
9. Escreva uma função que receba o valor de x e calcule e devolva o valor de $S = 1/4 + 2/9 + 3/16 + \dots + x/(x+1)^2$. Se a entrada for 5 a saída deverá ser 0.958611111.
10. Escreva uma função que receba uma lista de números e devolva o menor deles. Se a entrada for 33, 22, -10, -4 e 33 a saída deve ser -10.



408-75941 - e /nt

Parâmetros opcionais Seja uma função para sublinhar algo. Poderia ser

```
def sublinha():  
    print("_"*30)
```

Entretanto, nesta função tanto o caracter sublinha (`_`) quanto o valor 30 são fixos. Para permitir alterar isso, pode-se fazer:

```
def sublinha2(n=30, caractere="_"):  
    print(caractere*n)
```

Veja agora como a coisa é chamada

```
>>> def sublinha(n=30, cara="_"):  
    print(cara*n)  
>>> sublinha()
```

```
>>> sublinha(n=10)  
-----
```

```
>>> sublinha(cara='$')  
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
```

Funções em Python Além das centenas de funções pré-existentes em Python (print, min, max, sum, len, int, float, input...) a grande vantagem do ambiente é que você pode definir as funções que precisar para o seu trabalho. Depois que uma função foi definida ela passa a se comportar como se fosse (e é) uma função primitiva do Python.

Para definir uma nova função, usa-se o comando `def`, veja:

```
def nome(parâmetros):  
    comandos
```

Após a palavra `def` deve vir o nome (único) pelo qual a função será conhecida. Não pode ser uma palavra reservada do Python. Logo depois do nome, é obrigatório um "abre parênteses". A seguir uma quantidade opcional de parâmetros, que são como se fossem nomes de variáveis. Depois desses parâmetros vem o fecha parênteses. Se não houver parâmetro, depois do nome da função vem abre e fecha parênteses. Termina o cabeçalho o caráter "dois pontos" (:). Ele inicia um bloco que limita a definição da função.

Diferentemente do que aconteceu até agora, as linhas que fazem parte da função, não são executadas agora. Elas ficam armazenadas e serão executadas apenas quando a função agora definida, for chamada (executada).

A razão para definir uma função pode ser:

- * Para repetir o mesmo processamento em diversos locais de um programa.
- * Para encapsular um processamento específico.
- * Para comerciar ou apenas trocar algoritmos.
- * Para modularizar um grande processamento, permitindo que o mesmo seja desenvolvido por diversas pessoas.
- * Para poder manusear a complexidade crescente.

A função pode retornar algum resultado para quem a chama. Por exemplo, a função `len(lista)` retorna um valor que corresponde ao comprimento da lista que é passada como parâmetro.

Veja um exemplo de tudo que se disse

```
def maior(a):  
    #1  
    i=len(a)-1 #2  
    mai=-99999 #3  
    while i>= 0:  
        #4  
        if a[i]>mai:  
            #5  
            mai=a[i] #6  
        i=i-1 #7  
    return mai #8  
    pass #9  
print(maior([1,5,7,2,3,17,5])) #10
```

Na linha #1 define-se a função de nome `maior` e se diz que ela deverá ser chamada com um único parâmetro. Quem fez a função sabe que este parâmetro único deve ser uma lista formada por números. Na linha #2 cria-se a variável `i` e atribui-se a ela o valor do tamanho da lista parâmetro menos 1. Como se sabe tal índice acessa o último elemento da lista. Na linha #3 define-se uma outra variável de controle apta a guardar o maior valor da lista até aquele momento. Como ainda nenhum item foi processado ela vale $-\infty$. Na linha #4 define-se que `i` vai variar até ter um valor negativo. Na linha #5 se o elemento de ordem `i` na lista for maior do que a variável `mai`, `mai` será atualizada com este novo valor. Na linha #7 que já está fora da indentação do `if`, mas ainda dentro da indentação do `while` o índice `i` é recuado uma posição e graças ao `while` há um retorno à linha #4. Finalmente, quando o ciclo `while` acabar, vai-se para a linha #8 que simplesmente devolve o valor calculado a quem chamou a função `maior` e o processamento desta função está encerrado. A linha #9 não é executada (de qualquer maneira `pass` não faz nada mesmo), mas lembre-se QUALQUER coisa colocada após um `return` é meramente decorativo, pois nunca vai ser executado. Na linha #10, já fora da função `maior` (note a indentação) a mesma vai ser executada. Veja que preciso colocar um `print` para externalizar o resultado da função. Se não houver `print`, a função vai ser executada, mas nada será visto de fora.

O parâmetro é uma lista ? Se o parâmetro for uma lista de 4 números, ele será um único parâmetro. Estruturalmente é diferente de 4 parâmetros escalares, que no fundo são uma lista de 4 números. Veja

```
def f1(L4):  
    print(L4[0], L4[1], L4[2], L4[3])  
# ----- OU -----  
def f2(a,b,c,d):  
    print(a, b, c, d)  
f1([1,2,3,4])  
f2(1,2,3,4)
```

Qual estratégia usar ao modularizar um problema e definir quantas e quais funções devem ser construídas é uma arte: a bússola nesta tarefa é a melhor maneira de reaproveitar código e de evitar a reinvenção da roda.

Variáveis locais e globais Como regra geral todas as variáveis definidas dentro de uma função (como `i` e `maior` no exemplo aí acima), são chamadas **locais** à função e isto significa que elas nunca sobrevivem ao tempo de execução da função: não existem antes da função e deixam de existir quando a função acaba. Desta maneira não podem ter seu valor consultado e/ou alterado por alguém externo à função.

Este fato permite que duas funções distintas tenham variáveis com o mesmo nome (como é muito comum um índice chamado `i`) sem que um atrapalhe o outro. Desta maneira a única forma de ceder e receber valores entre funções é através de parâmetros e de retornos.

Já uma variável que precise ser vista e alterada por outros é conhecida como variável **GLOBAL**. Ela deve ser definida dentro de um módulo, mas **FORA** de uma função. Agora ela pode ser manuseada por todas as funções desse módulo e por todos os módulos que importam o módulo que a definiu.

Para que uma função possa alterar uma variável global, a mesma deve ser definida como global usando-se o comando **GLOBAL** como linha seguinte à definição.

Veja um exemplo:

```
xu = 77  
def fun():  
    xu = 2  
    print(xu)  
print(xu) # 77 é impresso  
fun() # 2 é impresso  
print(xu) # 77 é impresso  
def alva():  
    global xu  
    xu=2  
alva()  
print(xu) # 2 é impresso
```

As boas práticas de programação sugerem não usar variáveis globais. Esta prática viola o encapsulamento e é fonte frequente de bugs de programação.

Seus módulos podem ser importados Organizando seu trabalho direitinho, você pode modularizar seus programas e importá-los quando necessário, acompanhe

```
#----- beta.py -----  
def alfa(x):  
    return x**3  
#-----  
>>> import beta  
>>> print(beta.alfa(4))  
#-----  
>>> from beta import *  
>>> print(alfa(4))
```

👉 Para você fazer

A seguir uma lista de funções que você deve programar em Python. Imprima as funções e devolva-as impressas junto com esta folha. Não esqueça de identificar com seu nome e o código 19 limite entrega 10/10/24

1. Escreva uma função que receba dois números e devolva `True` se tais números puderem indicar um dia e mês e `False` senão. Por exemplo, se a entrada for 23,11 a saída deverá ser `True`, idem se for 1,1. Se a entrada for 33,5 a saída deverá ser `False`.

2. Escreva uma função que receba um string contendo uma frase e devolva a quantidade de vogais que foram usadas na frase. Se a entrada for "Ivo viu a uva" a saída deverá ser 7.
3. Escreva uma função que receba dois números: o preço do álcool e o preço da gasolina. Sabendo que um carro em particular faz um certo trajeto gastando 10l de álcool e 7l de gasolina (o mesmo trajeto), a função deve recomendar o combustível mais barato, a saber `ALCOOL` ou `GASOLINA`. Por exemplo, se $A=2.90$ R\$ e $G=4.10$ R\$, fazendo as contas, a gasolina é mais eficiente e a função deve recomendá-la.
4. Escreva uma função que receba dois números que descrevem o desempenho de alguém ao cursar a disciplina de `MÉTODOS NUMÉRICOS`. O primeiro número é a frequência ($0.0 \leq f \leq 100.0$) em percentagem e o segundo número é a nota final obtida ($0.0 \leq NF \leq 10.0$). Deve imprimir a mensagem `APROVADO` ou `REPROVADO` a depender das regras da UFPR para isso. Por exemplo, se $f = 66$ e $NF = 7.1$ a mensagem deve ser `REPROVADO`.
5. Escreva uma função que receba uma lista com 3 valores numéricos: `A`, `B` e `C` e calcule as raízes de $Ax^2 + Bx + c = 0$, usando a fórmula de Baskara. Se a entrada for 1, -5 e 6 a resposta deverá ser 2 e 3.
6. Escreva uma função que receba uma lista de números calcule e devolva a média dos elementos da lista. Se a entrada for 8 7 4 19 21 a saída deverá ser 11.8.
7. Escreva uma função que receba dois números e devolva `True` se tais números puderem indicar uma hora e minuto e `False` senão. Por exemplo, se a entrada for 23,59 a saída deverá ser `True`, idem se for 0,0. Se a entrada for 33,5 a saída deverá ser `False`.
8. Escreva uma função que receba um número inteiro `x` e devolva `True` se ele for primo e `False` senão. Um número é primo quando seus únicos divisores são ele mesmo e a unidade. Se a entrada for 17 a saída deverá ser `True`.
9. Escreva uma função que receba uma lista de números contendo uma distribuição qualquer e devolva a distribuição acumulada. Por exemplo, se a entrada for 1, 5, 7, 10, 21 a saída deverá ser 1, 6, 13, 23, 44.
10. Escreva uma função que receba uma lista de números e devolva o maior deles. Se a entrada for 33, 22, -10, -4 e 33 a saída deve ser 33.



408-75958 - e /nt

Parâmetros opcionais Seja uma função para sublinhar algo. Poderia ser

```
def sublinha():  
    print("_"*30)  
Entretanto, nesta função tanto o caracter sublinha (_) quanto o valor 30 são fixos. Para permitir alterar isso, pode-se fazer:  
def sublinha2(n=30, caractere="_"):  
    print(caractere*n)  
Veja agora como a coisa é chamada  
>>> def sublinha(n=30, cara="_"):  
    print(cara*n)  
>>> sublinha()  
-----  
>>> sublinha(n=10)  
-----  
>>> sublinha(cara='$')  
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
```

Funções em Python Além das centenas de funções pré-existentes em Python (print, min, max, sum, len, int, float, input...) a grande vantagem do ambiente é que você pode definir as funções que precisar para o seu trabalho. Depois que uma função foi definida ela passa a se comportar como se fosse (e é) uma função primitiva do Python.

Para definir uma nova função, usa-se o comando `def`, veja:

```
def nome(parâmetros):  
    comandos
```

Após a palavra `def` deve vir o nome (único) pelo qual a função será conhecida. Não pode ser uma palavra reservada do Python. Logo depois do nome, é obrigatório um "abre parênteses". A seguir uma quantidade opcional de parâmetros, que são como se fossem nomes de variáveis. Depois desses parâmetros vem o fecha parênteses. Se não houver parâmetro, depois do nome da função vem abre e fecha parênteses. Termina o cabeçalho o caráter "dois pontos" (:). Ele inicia um bloco que limita a definição da função.

Diferentemente do que aconteceu até agora, as linhas que fazem parte da função, não são executadas agora. Elas ficam armazenadas e serão executadas apenas quando a função agora definida, for chamada (executada).

A razão para definir uma função pode ser:

- * Para repetir o mesmo processamento em diversos locais de um programa.
- * Para encapsular um processamento específico.
- * Para comercializar ou apenas trocar algoritmos.
- * Para modularizar um grande processamento, permitindo que o mesmo seja desenvolvido por diversas pessoas.
- * Para poder manusear a complexidade crescente.

A função pode retornar algum resultado para quem a chama. Por exemplo, a função `len(lista)` retorna um valor que corresponde ao comprimento da lista que é passada como parâmetro.

Veja um exemplo de tudo que se disse

```
def maior(a):  
    #1  
    i=len(a)-1  
    #2  
    mai=-99999  
    #3  
    while i>= 0:  
        #4  
        if a[i]>mai:  
            #5  
            mai=a[i]  
            #6  
            i=i-1  
            #7  
    return mai  
    #8  
    pass  
    #9  
print(maior([1,5,7,2,3,17,5]))
```

Na linha #1 define-se a função de nome `maior` e se diz que ela deverá ser chamada com um único parâmetro. Quem fez a função sabe que este parâmetro único deve ser uma lista formada por números. Na linha #2 cria-se a variável `i` e atribui-se a ela o valor do tamanho da lista parâmetro menos 1. Como se sabe tal índice acessa o último elemento da lista. Na linha #3 define-se uma outra variável de controle apta a guardar o maior valor da lista até aquele momento. Como ainda nenhum item foi processado ela vale $-\infty$. Na linha #4 define-se que `i` vai variar até ter um valor negativo. Na linha #5 se o elemento de ordem `i` na lista for maior do que a variável `mai`, `mai` será atualizada com este novo valor. Na linha #7 que já está fora da indentação do `if`, mas ainda dentro da indentação do `while` o índice `i` é recuado uma posição e graças ao `while` há um retorno à linha #4. Finalmente, quando o ciclo `while` acabar, vai-se para a linha #8 que simplesmente devolve o valor calculado a quem chamou a função `maior` e o processamento desta função está encerrado. A linha #9 não é executada (de qualquer maneira `pass` não faz nada mesmo), mas lembre-se QUALQUER coisa colocada após um `return` é meramente decorativo, pois nunca vai ser executado. Na linha #10, já fora da função `maior` (note a indentação) a mesma vai ser executada. Veja que preciso colocar um `print` para externalizar o resultado da função. Se não houver `print`, a função vai ser executada, mas nada será visto de fora.

O parâmetro é uma lista ? Se o parâmetro for uma lista de 4 números, ele será um único parâmetro. Estruturalmente é diferente de 4 parâmetros escalares, que no fundo são uma lista de 4 números. Veja

```
def f1(L4):  
    print(L4[0], L4[1], L4[2], L4[3])  
# ----- OU -----  
def f2(a,b,c,d):  
    print(a, b, c, d)  
f1([1,2,3,4])  
f2(1,2,3,4)
```

Qual estratégia usar ao modularizar um problema e definir quantas e quais funções devem ser construídas é uma arte: a bússola nesta tarefa é a melhor maneira de reaproveitar código e de evitar a reinvenção da roada.

Variáveis locais e globais Como regra geral todas as variáveis definidas dentro de uma função (como `i` e `maior` no exemplo aí acima), são chamadas **locais** à função e isto significa que elas nunca sobrevivem ao tempo de execução da função: não existem antes da função e deixam de existir quando a função acaba. Desta maneira não podem ter seu valor consultado e/ou alterado por alguém externo à função.

Este fato permite que duas funções distintas tenham variáveis com o mesmo nome (como é muito comum um índice chamado `i`) sem que um atrapalhe o outro. Desta maneira a única forma de ceder e receber valores entre funções é através de parâmetros e de retornos.

Já uma variável que precise ser vista e alterada por outros é conhecida como variável **GLOBAL**. Ela deve ser definida dentro de um módulo, mas FORA de uma função. Agora ela pode ser manuseada por todas as funções desse módulo e por todos os módulos que importam o módulo que a definiu.

Para que uma função possa alterar uma variável global, a mesma deve ser definida como global usando-se o comando `GLOBAL` como linha seguinte à definição.

Veja um exemplo:

```
xu = 77  
def fun():  
    xu = 2  
    print(xu)  
print(xu) # 77 é impresso  
fun() # 2 é impresso  
print(xu) # 77 é impresso  
def alva():  
    global xu  
    xu=2  
alva()  
print(xu) # 2 é impresso
```

As boas práticas de programação sugerem não usar variáveis globais. Esta prática viola o encapsulamento e é fonte frequente de bugs de programação.

Seus módulos podem ser importados Organizando seu trabalho direitinho, você pode modularizar seus programas e importá-los quando necessário, acompanhe

```
#----- beta.py -----  
def alfa(x):  
    return x**3  
#-----  
>>> import beta  
>>> print(beta.alfa(4))  
#-----  
>>> from beta import *  
>>> print(alfa(4))
```

☞ Para você fazer

A seguir uma lista de funções que você deve programar em Python. Imprima as funções e devolva-as impressas junto com esta folha. Não esqueça de identificar com seu nome e o código 20 limite entrega 10/10/24

1. Escreva uma função que receba uma lista de números e retorne a sua média. Por exemplo, se a entrada for 1, 2, 3, 4 a saída deverá ser 2.5.
2. Um plano de telefonia celular tem uma taxa de R\$ 35 por mês. O plano inclui 400 minutos gratuitos e taxa de 10 centavos para cada minuto adicional utilizado. Escreva

uma função que indique o custo mensal em função do número de minutos utilizados. Por exemplo se a função for chamada com 410, deve responder 36 R\$.

3. Escreva uma função que receba uma lista com 3 valores numéricos: A, B e C e calcule as raízes de $Ax^2 + Bx + c = 0$, usando a fórmula de Baskara. Se a entrada for 1, -5 e 6 a resposta deverá ser 2 e 3.
4. Escreva uma função que receba um string contendo uma frase e devolva a quantidade de palavras que formam a frase. Se a entrada for "IVO VIU A UVA" a saída deverá ser 4.
5. Escreva uma função que receba uma lista de números e imprima-a em ordem reversa (de trás para a frente). Se a entrada for 4, 7, -3, 22, 14, a impressão deverá ser de 14, 22, -3, 7 e 4.
6. Escreva uma função que receba um número inteiro e devolva a soma de todos os divisores deste número. Se a entrada for 15 a saída deverá ser 24.
7. Escreva uma função que receba uma lista de números calcule e devolva a média dos elementos da lista. Se a entrada for 8 7 4 19 21 a saída deverá ser 11.8.
8. Escreva uma função que receba o valor de x e calcule e devolva o valor de $S = 1/4 + 2/9 + 3/16 + \dots + x/(x+1)^2$. Se a entrada for 5 a saída deverá ser 0.958611111.
9. Escreva uma função que receba um string contendo uma frase e devolva a quantidade de vogais que foram usadas na frase. Se a entrada for "Ivo viu a uva" a saída deverá ser 7.
10. Escreva uma função que receba um número inteiro x e calcule e devolva a soma dos múltiplos de 13 menores que x . Se a entrada for 100, a saída deverá ser 364 que é igual à soma de 13 26 39 52 65 78 91.



408-75965 - e /nt

Parâmetros opcionais Seja uma função para sublinhar algo. Poderia ser

```
def sublinha():  
    print("#*30")  
Entretanto, nesta função tanto o caracter sublinha (_) quanto o valor 30 são fixos. Para permitir alterar isso, pode-se fazer:  
def sublinha2(n=30, caractere="#"):  
    print(caractere*n)  
Veja agora como a coisa é chamada  
>>> def sublinha(n=30, cara="#"):  
    print(cara*n)  
>>> sublinha()  
-----  
>>> sublinha(n=10)  
-----  
>>> sublinha(cara='$')  
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
```

Funções em Python Além das centenas de funções pré-existentes em Python (print, min, max, sum, len, int, float, input...) a grande vantagem do ambiente é que você pode definir as funções que precisar para o seu trabalho. Depois que uma função foi definida ela passa a se comportar como se fosse (e é) uma função primitiva do Python.

Para definir uma nova função, usa-se o comando `def`, veja:

```
def nome(parâmetros):  
    comandos
```

Após a palavra `def` deve vir o nome (único) pelo qual a função será conhecida. Não pode ser uma palavra reservada do Python. Logo depois do nome, é obrigatório um "abre parênteses". A seguir uma quantidade opcional de parâmetros, que são como se fossem nomes de variáveis. Depois desses parâmetros vem o fecha parênteses. Se não houver parâmetro, depois do nome da função vem abre e fecha parênteses. Termina o cabeçalho o caráter "dois pontos" (:). Ele inicia um bloco que limita a definição da função.

Diferentemente do que aconteceu até agora, as linhas que fazem parte da função, não são executadas agora. Elas ficam armazenadas e serão executadas apenas quando a função agora definida, for chamada (executada).

A razão para definir uma função pode ser:

- * Para repetir o mesmo processamento em diversos locais de um programa.
- * Para encapsular um processamento específico.
- * Para comerciar ou apenas trocar algoritmos.
- * Para modularizar um grande processamento, permitindo que o mesmo seja desenvolvido por diversas pessoas.
- * Para poder manusear a complexidade crescente.

A função pode retornar algum resultado para quem a chama. Por exemplo, a função `len(lista)` retorna um valor que corresponde ao comprimento da lista que é passada como parâmetro.

Veja um exemplo de tudo que se disse

```
def maior(a):  
    i=len(a)-1 #1  
    mai=-99999 #2  
    while i>= 0: #3  
        if a[i]>mai: #4  
            mai=a[i] #5  
        i=i-1 #6  
    return mai #7  
    pass #8  
print(maior([1,5,7,2,3,17,5])) #9
```

Na linha #1 define-se a função de nome `maior` e se diz que ela deverá ser chamada com um único parâmetro. Quem fez a função sabe que este parâmetro único deve ser uma lista formada por números. Na linha #2 cria-se a variável `i` e atribui-se a ela o valor do tamanho da lista parâmetro menos 1. Como se sabe tal índice acessa o último elemento da lista. Na linha #3 define-se uma outra variável de controle apta a guardar o maior valor da lista até aquele momento. Como ainda nenhum item foi processado ela vale $-\infty$. Na linha #4 define-se que `i` vai variar até ter um valor negativo. Na linha #5 se o elemento de ordem `i` na lista for maior do que a variável `mai`, `mai` será atualizada com este novo valor. Na linha #7 que já está fora da indentação do `if`, mas ainda dentro da indentação do `while` o índice `i` é recuado uma posição e graças ao `while` há um retorno à linha #4. Finalmente, quando o ciclo `while` acabar, vai-se para a linha #8 que simplesmente devolve o valor calculado a quem chamou a função `maior` e o processamento desta função está encerrado. A linha #9 não é executada (de qualquer maneira `pass` não faz nada mesmo), mas lembre-se QUALQUER coisa colocada após um `return` é meramente decorativo, pois nunca vai ser executado. Na linha #10, já fora da função `maior` (note a indentação) a mesma vai ser executada. Veja que preciso colocar um `print` para externalizar o resultado da função. Se não houver `print`, a função vai ser executada, mas nada será visto de fora.

O parâmetro é uma lista ? Se o parâmetro for uma lista de 4 números, ele será um único parâmetro. Estruturalmente é diferente de 4 parâmetros escalares, no fundo são uma lista de 4 números. Veja

```
def f1(L4):  
    print(L4[0], L4[1], L4[2], L4[3])  
# ---- ou ----  
def f2(a,b,c,d):  
    print(a, b, c, d)  
f1([1,2,3,4])  
f2(1,2,3,4)
```

Qual estratégia usar ao modularizar um problema e definir quantas e quais funções devem ser construídas é uma arte: a bússola nesta tarefa é a melhor maneira de reaproveitar código e de evitar a reinvenção da roda.

Variáveis locais e globais Como regra geral todas as variáveis definidas dentro de uma função (como `i` e `maior` no exemplo aí acima), são chamadas **locais** à função e isto significa que elas nunca sobrevivem ao tempo de execução da função: não existem antes da função e deixam de existir quando a função acaba. Desta maneira não podem ter seu valor consultado e/ou alterado por alguém externo à função.

Este fato permite que duas funções distintas tenham variáveis com o mesmo nome (como é muito comum um índice chamado `i`) sem que um atrapalhe o outro. Desta maneira a única forma de ceder e receber valores entre funções é através de parâmetros e de retornos.

Já uma variável que precise ser vista e alterada por outros é conhecida como variável **GLOBAL**. Ela deve ser definida dentro de um módulo, mas FORA de uma função. Agora ela pode ser manuseada por todas as funções desse módulo e por todos os módulos que importam o módulo que a definiu.

Para que uma função possa alterar uma variável global, a mesma deve ser definida como global usando-se o comando **GLOBAL** como linha seguinte à definição.

Veja um exemplo:

```
xu = 77  
def fun():  
    xu = 2  
    print(xu)  
print(xu) # 77 é impresso  
fun() # 2 é impresso  
print(xu) # 77 é impresso  
def alva():  
    global xu  
    xu=2  
alva()  
print(xu) # 2 é impresso
```

As boas práticas de programação sugerem não usar variáveis globais. Esta prática viola o encapsulamento e é fonte frequente de bugs de programação.

Seus módulos podem ser importados Organizando seu trabalho direitinho, você pode modularizar seus programas e importá-los quando necessário, acompanhe

```
#----- beta.py -----  
def alfa(x):  
    return x**3  
#-----  
>>> import beta  
>>> print(beta.alfa(4))  
#-----  
>>> from beta import *  
>>> print(alfa(4))
```

👉 Para você fazer

A seguir uma lista de funções que você deve programar em Python. Imprima as funções e devolva-as impressas junto com esta folha. Não esqueça de identificar com seu nome e o código 21 limite entrega 10/10/24

1. Escreva uma função que receba um string contendo uma frase e devolva a quantidade de vogais que foram usadas na frase. Se a entrada for "Ivo viu a uva" a saída deverá ser 7.

2. Escreva uma função que receba uma lista de números e retorne a sua média. Por exemplo, se a entrada for 1, 2, 3, 4 a saída deverá ser 2.5.
3. Escreva uma função que receba um valor numérico indicando o lado de um quadrado, medido em metros e devolva o valor do perímetro e da área do quadrado. Se a entrada for 1, a saída deve ser 4 e 1.
4. Escreva uma função que receba um número inteiro `x` e calcule e devolva a soma dos múltiplos de 13 menores que `x`. Se a entrada for 100, a saída deverá ser 364 que é igual à soma de 13 26 39 52 65 78 91.
5. Escreva uma função que receba um string contendo uma frase e devolva a quantidade de palavras que formam a frase. Se a entrada for "IVO VIU A UVA" a saída deverá ser 4.
6. Escreva uma função que receba um número inteiro `x` e devolva `True` se ele for primo e `False` senão. Um número é primo quando seus únicos divisores são ele mesmo e a unidade. Se a entrada for 17 a saída deverá ser `True`.
7. Um plano de telefonia celular tem uma taxa de R\$ 35 por mês. O plano inclui 400 minutos gratuitos e taxa de 10 centavos para cada minuto adicional utilizado. Escreva uma função que indique o custo mensal em função do número de minutos utilizados. Por exemplo se a função for chamada com 410, deve responder 36 R\$.
8. Uma empresa de eletricidade cobra de seus clientes uma taxa base de R\$ 10 mensais mais seis centavos por quilowatt-hora para os primeiros 1200 quilowatt-hora e 7 centavos por quilowatt-hora para todo uso acima de 1.200. Escreva uma função que receba o consumo e indique o valor a ser cobrado. Por exemplo, para um consumo de 1000 Kwh, deve cobrar 10+60=70 R\$.
9. Uma janela normanda tem o formato de um quadrado sobre o qual se coloca um semicírculo. Escreva uma função que receba o perímetro da janela e indique qual o lado do quadrado no qual ela está inserida. Por exemplo, se o perímetro da janela é de 10m, o lado do quadrado é 2,18m.
10. Escreva uma função que receba dois números e devolva `True` se tais números puderem indicar uma hora e minuto e `False` senão. Por exemplo, se a entrada for 23,59 a saída deverá ser `True`, idem se for 0,0. Se a entrada for 33,5 a saída deverá ser `False`.



408-75972 - e /nt

Parâmetros opcionais Seja uma função para sublinhar algo. Poderia ser

```
def sublinha():  
    print("_"*30)  
Entretanto, nesta função tanto o caracter sublinha (_) quanto o valor 30 são fixos. Para permitir alterar isso, pode-se fazer:  
def sublinha2(n=30, caractere="_"):  
    print(caractere*n)  
Veja agora como a coisa é chamada  
>>> def sublinha(n=30, cara="_"):  
    print(cara*n)  
>>> sublinha()  
-----  
>>> sublinha(n=10)  
-----  
>>> sublinha(cara='$')  
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
```

Listagem de todos os problemas de 932

1. Escreva uma função que receba uma lista de números e retorne a sua média. Por exemplo, se a entrada for 1, 2, 3, 4 a saída deverá ser 2.5. trivial
2. Escreva uma função que receba um parâmetro numérico e retorne a palavra PAR ou a palavra IMPAR se o parâmetro for par ou impar. Por exemplo, se a entrada for 3, a saída deverá ser IMPAR. trivial uhuhuh uhuhuhu uhuhuh
3. Escreva uma função que receba uma lista formada por 3 números (se vierem menos ou mais deve dar um erro e encerrar) e deve imprimir SIM se os números forem lados de um triângulo retângulo e NÃO se não puderem ser. Por exemplo, se a entrada for 3,5,4 a saída deverá ser SIM. Se a entrada for 1,10,100 a saída deverá ser NÃO. Use o teorema de pitágoras
4. Escreva uma função que receba uma lista com 3 valores numéricos: A, B e C e calcule as raízes de $Ax^2 + Bx + c = 0$, usando a fórmula de Báskara. Se a entrada for 1, -5 e 6 a resposta deverá ser 2 e 3. trivial
5. Escreva uma função que receba um valor numérico indicando o lado de um quadrado, medido em metros e devolva o valor do perímetro e da área do quadrado. Se a entrada for 1, a saída deve ser 4 e 1.
6. Escreva uma função que receba uma lista de números e devolva o maior deles. Se a entrada for 33, 22, -10, -4 e 33 a saída deve ser 33.
7. Escreva uma função que receba uma lista de números e devolva o menor deles. Se a entrada for 33, 22, -10, -4 e 33 a saída deve ser -10.
8. Escreva uma função que receba uma lista de números e devolva a diferença entre o maior e o menor deles. Se a entrada for 33, 22, -10, -4 e 33 a saída deve ser 43.
9. Escreva uma função que receba um número inteiro e devolva uma lista com os divisores deste número. Se a entrada for 16, a saída deverá ser 1,2,4,8,16.
10. Escreva uma função que receba um número inteiro e devolva a soma de todos os divisores deste número. Se a entrada for 15 a saída deverá ser 24.
11. Escreva uma função que receba 2 valores: o peso e a altura de alguém e devolva o IMC (Índice de Massa Corporal): $IMC = peso \div (altura)^2$. Peso em Kg e altura em m. Se a entrada for 80, 1.80 a saída deverá ser 24.69.
12. Escreva uma função que receba o valor de x e calcule e devolva o valor de $S = 1/2 + 2/3 + 3/4 + \dots + x/x + 1$. Se a entrada for 5 a saída deverá ser 3.55.
13. Escreva uma função que receba o valor de x e calcule e devolva o valor de $S = 1/4 + 2/9 + 3/16 + \dots + x/(x + 1)^2$. Se a entrada for 5 a saída deverá ser 0.958611111.
14. Escreva uma função que receba um número inteiro x e calcule e devolva a soma dos múltiplos de 13 menores que x . Se a entrada for 100, a saída deverá ser 364 que é igual à soma de 13 26 39 52 65 78 91.
15. Escreva uma função que receba um número inteiro x e devolva *True* se ele for primo e *False* senão. Um número é primo quando seus únicos divisores são ele mesmo e a unidade. Se a entrada for 17 a saída deverá ser *True*.
16. Escreva uma função que receba uma lista de números calcule e devolva a média dos elementos da lista. Se a entrada for 8 7 4 19 21 a saída deverá ser 11.8.
17. Escreva uma função que receba uma lista de números e imprima-a em ordem reversa (de trás para a frente). Se a entrada for 4, 7, -3, 22, 14, a impressão deverá ser de 14, 22, -3, 7 e 4.
18. Escreva uma função que receba um string contendo uma frase e devolva a quantidade de palavras que formam a frase. Se a entrada for "IVO VIU A UVA" a saída deverá ser 4.
19. Escreva uma função que receba um string contendo uma frase e devolva a quantidade de palavras masculinas que formam a frase. Por hipótese, as palavras masculinas são as que terminam em "O" ou "o". Se a entrada for "O RATO ROEU A ROUPA DO REI" a saída deverá ser 3.
20. Escreva uma função que receba um string contendo uma frase e devolva a quantidade de vogais que foram usadas na frase. Se a entrada for "Ivo viu a uva" a saída deverá ser 7.
21. Escreva uma função que receba uma lista de números e devolva *True* se a lista estiver em ordem crescente e *False* senão. Uma lista X_i está em ordem crescente se $x_{i+1} \geq x_i$. Se a entrada for 1,2,2,2,2,3 a saída deverá ser *True*.
22. Escreva uma função que receba uma lista de números e devolva a quantidade de números que são menores do que seus vizinhos. Os extremos da lista não devem ser testados pois lhes falta algum vizinho. Se a entrada for 1,4,2,8,22,50,100 a saída deverá ser 1.

23. Escreva uma função que receba uma lista de números positivos e negativos misturados e devolva duas listas: a primeira contendo apenas os positivos e a segunda apenas os negativos, na mesma ordem em que eles apareceram na entrada. Se a entrada for -2, 4, -4, -6, -11, 4, 18 as saídas deverão ser -2, -4, -6, -11 e 4, 4, 18.
24. Escreva uma função que receba uma lista de números contendo uma distribuição qualquer e devolva a distribuição acumulada. Por exemplo, se a entrada for 1, 5, 7, 10, 21 a saída deverá ser 1, 6, 13, 23, 44.
25. Escreva uma função que receba 2 horários (no formato HH, MM) indicando o início e o fim de uma determinada tarefa. A função deve devolver a duração em minutos da tarefa. Se a entrada for 1,10 e 2,0 a saída deverá ser 50.
26. Escreva uma função que receba dois números e devolva *True* se tais números puderem indicar uma hora e minuto e *False* senão. Por exemplo, se a entrada for 23,59 a saída deverá ser *True*, idem se for 0,0. Se a entrada for 33,5 a saída deverá ser *False*.
27. Escreva uma função que receba dois números e devolva *True* se tais números puderem indicar um dia e mês e *False* senão. Por exemplo, se a entrada for 23,11 a saída deverá ser *True*, idem se for 1,1. Se a entrada for 33,5 a saída deverá ser *False*.
28. Escreva uma função que receba a altura e o peso de uma pessoa. Deve calcular o IMC – índice de massa corporal ($IMV = peso/altura^2$) medidos em kg e m. A seguir deve emitir mensagem a depender do IMC. $IMC < 25$ =adequado; $IMC < 30$ =sobrepeso; $IMC < 35$ =obesidade e $IMC \geq 35$ =obesidade severa. Por exemplo, se o sujeito pesa 110Kg e tem 1.70m de altura seu IMC é de 38.06. Logo o diagnóstico é OBESIDADE SEVERA.
29. Escreva uma função que receba dois números: o preço do álcool e o preço da gasolina. Sabendo que um carro em particular faz um certo trajeto gastando 10l de álcool e 7l de gasolina (o mesmo trajeto), a função deve recomendar o combustível mais barato, a saber ALCOOL ou GASOLINA. Por exemplo, se A=2.90 R\$ e G=4.10 R\$, fazendo as contas, a gasolina é mais eficiente e a função deve recomendá-la.
30. Escreva uma função que receba dois números que descrevem o desempenho de alguém ao cursar a disciplina de MÉTODOS NUMÉRICOS. O primeiro número é a frequência ($0.0 \leq f \leq 100.0$) em porcentagem e o segundo número é a nota final obtida ($0.0 \leq NF \leq 10.0$). Deve imprimir a mensagem APROVADO ou REPROVADO a depender das regras da UFPR para isso. Por exemplo, se $f = 66$ e $NF = 7.1$ a mensagem deve ser REPROVADO.
31. Um plano de telefonia celular tem uma taxa de R\$ 35 por mês. O plano inclui 400 minutos gratuitos e taxa de 10 centavos para cada minuto adicional utilizado. Escreva uma função que indique o custo mensal em função do número de minutos utilizados. Por exemplo se a função for chamada com 410, deve responder 36 R\$.
32. Em uma certa província a velocidade máxima permitida em estradas é de 100 km por hora e a velocidade mínima é de 50 km por hora. A multa por violar estes limites é de R\$ 10 para cada km por hora acima da velocidade máxima ou abaixo da velocidade mínima. Escreva uma função que receba a velocidade de condução e indique quanto deve ser a multa se é que ela deva existir. Em caso negativo, retornar 0. Por exemplo, se for chamada com 75 km/h, deve retornar 0. Se for chamada com 110 ou com 40 Km/h deve retornar 100,00.
33. Uma empresa de eletricidade cobra de seus clientes uma taxa base de R\$ 10 mensais mais seis centavos por quilowatt-hora para os primeiros 1200 quilowatt-hora e 7 centavos por quilowatt-hora para todo uso acima de 1.200. Escreva uma função que receba o consumo e indique o valor a ser cobrado. Por exemplo, para um consumo de 1000 Kwh, deve cobrar $10+60=70$ R\$.
34. Uma janela normanda tem o formato de um quadrado sobre o qual se coloca um semicírculo. Escreva uma função que receba o perímetro da janela e indique qual o lado do quadrado no qual ela está inserida. Por exemplo, se o perímetro da janela é de 10m, o lado do quadrado é 2,18m.