Funções

Observe a sintaxe abaixo:

```
def nome_função(argumentos_opcionais):
    <instruções (corpo da função)>
```

- começa sempre com a palavra-chave *def* (de define);
- depois de *def* passamos o nome da função (as regras para nomear funções são exatamente as mesmas que para nomear variáveis);
- em seguida ao nome da função, há um par de parênteses obrigatórios, são os argumentos opcionais da função;
- a linha deve ser terminada com dois pontos;
- a linha logo após *def* inicia o corpo da função - um par (pelo menos um) de instruções necessariamente aninhadas, que serão executadas sempre que a função for chamada.

função sem argumentos

In [ ]:
```
# coloque seu código aqui
```

O que acontece quando você tenta invocar uma função antes de defini-la? Exemplo:

In [ ]:
```
# coloque seu código aqui

def oi():
    print("Bom dia!")
```

O que acontecerá quando você executar o código abaixo?

In [ ]:
```
# coloque seu código aqui

def oi():
    print("Bom dia")

oi("ok")
```

função com argumentos

In [ ]:
```
# coloque seu código aqui
def funçao(nome, lugar):
    print(f"ola {nome}: voce e de {lugar}?")

#chamando a funçao
funçao("maria","paris")
funçao("sao paulo","roberto")
```

```
ola maria: voce e de paris?
ola sao paulo: voce e de roberto?
```

argumentos posicionais

O Python oferece outra convenção para a passagem de argumentos, em que o significado do argumento é determinado por seu nome, e não por sua posição - é chamado de passagem de argumento de *palavra-chave*.
Portanto, no exemplo acima, o primeiro argumento é o `nome` e o segundo é o `lugar`.
Nesse caso, podemos usar a *palavra-chave* para determinar qual argumento queremo usar, independente da posição original.

In [ ]:
```
funçao(lugar= "rio de janeiro",nome="paulo")
```

```
ola paulo: voce e de rio de janeiro?
```

valor-padrão

Vimos aqui que a posição passou a ser irrelevante pq vc colocou os nomes de cada um dos argumentos.
Podemos especificar os *valores-padrão* dos argumentos caso eles não sejam fornecidos.
Nesse caso, os argumentos `valor` e `bonus` assumem os valores determinados na função toda vez que ele não for explicitamente fornecido na chamada da função.

In [ ]:
```
def comissao(valor=1000, bonus=50):
    total = valor + bonus
    print(f" o total da comissao sera de R${total:.2f}")

#chamando a funçao
comissao()
```

```
 o total da comissao sera de R$1050.00
```

In [ ]:
```
#chamando a funçao
comissao(200,20)
```

```
 o total da comissao sera de R$220.00
```

In [ ]:
```
#chamando a funçao
comissao(bonus=1000)
```

```
 o total da comissao sera de R$2000.00
```

In [ ]:
```
#chamando a funçao
comissao(2000)
```

```
 o total da comissao sera de R$2050.00
```

In [ ]:
```
#chamando a funçao
comissao(5000, bonus=80)
```

```
 o total da comissao sera de R$5080.00
```

In [ ]:
```
#chamando a funçao
comissao(valor=5000, 80)

# argumento posicional nao pode aparecer apos argumento de palavras-chave
```

```
  Cell In[39], line 2
    comissao(valor=5000, 80)
                         ^
SyntaxError: positional argument follows keyword argument
```

Recebendo valores de input

In [ ]:
```
# coloque seu código aqui
def endereço(rua,cidade,cep):
    print("seu endereço e:",rua,"-",cidade,"-",cep)

r = input("rua:")
cd = input("cidade:")
cp = input("cep:")
endereço(r,cd,cp)
```

```
seu endereço e: bartolomeu - diadema - 00000000
```

Retornando um resultado de uma função

```
def nome_função(argumentos_opcionais):
    <instruções (corpo da função)>
    return expression
```

In [ ]:
```
# coloque seu código aqui
def calcular(a,b):
    resultado = 0
    resultado = (a + b)**2
    return resultado

numero1 = int(input("digitando um numero inteiro:"))
numero2 = int(input("digitando um numero inteiro:"))

resultado = calcular(numero1,numero2)
print(resultado)
```

```
100
```

Essa função recebe dois argumentos e exibe o resultado. No entanto, assim que a função é encerrada, a variável `resultado` é deletada. O que acontece se tentarmos acessá-la?

In [ ]:
```
# coloque seu código aqui

print(resultado)
```

```
100
```

Para utilizar a variável devemos usar a palavra-chave `return` dentro de uma função.

In [ ]:
```
# coloque seu código aqui
```

In [ ]:
```
# coloque seu código aqui
```

Agora podemos fazer outras manipulações com base no resultado que obtivemos.

In [ ]:
```
# coloque seu código aqui
```

Retomando a função do cálculo da média aritmética:

In [ ]:
```
# coloque seu código aqui
def calcular_media(numeros):
    ' ...'
    ' calcula a media aritmetica '
    ' usando um lista '
    ' ... '

    total = 0
    for num in numeros:
        total += num

    mediaartm = total / len(numeros)
    return mediaartm

nlista = [10, 10, 15, 20, 40]
mediaartm = calcular_media(nlista)
print("a media e:",(mediaartm))
```

```
  Cell In[68], line 8
    total = 0
    ^
IndentationError: unexpected indent
```

Documentação - boas práticas - docstrings

In [ ]:
```
# coloque seu código aqui
```

In [ ]:
```
# documentação da função nativa len()
print(len.__doc__)
```