Funções

Na programação, funções são blocos de código que realizam determinadas tarefas que normalmente precisam ser executadas diversas vezes dentro de uma aplicação.

Quando surge essa necessidade, para que várias instruções não precisem ser repetidas, elas são agrupadas em uma função, à qual é dado um nome e que poderá ser chamada/executada em diferentes partes do programa.

In [1]:

```
def main():
    # comandos da função
    pass

main()
```

```
Primeira Execução
```

Criando Funções

A sintaxe da função é definida por três parte:

- **Nome:** como a chamamos
- **Parâmetros:** o que ela vai utilizar de variavel externa para executar o comportamento
- **Comportamento:** o que a função fará

No código abaixo temos um exemplo de declaração de função em Python

In [6]:

```
def nome(parametro):
    comportamento = ''
```

In [7]:

```
def ola(meu_nome):
    print("Olá,",meu_nome)
```

Essa função, de nome ola, tem como objetivo imprimir o nome que lhe é passado por parâmetro (também chamado de argumento). A palavra reservada **def**, na primeira linha, explicita a definição da função naquele ponto. Em seguida, entre parênteses, temos o parâmetro meu_nome. Ainda na mesma linha, observe a utilização dos dois pontos (:), que indicam que o código identado nas linhas abaixo faz parte da função que está sendo criada. Aqui, é importante ressaltar que, para respeitar a sintaxe da linguagem, a linha 2 está avançada em relação à linha 1.

Para executar a função, de forma semelhante ao que ocorre em outras linguagens, devemos simplesmente chamar seu nome e passar os parâmetros esperados entre parênteses, conforme o código a seguir.

In [8]:

```
ola('Felipe')
```

```
Olá, Felipe
```

Caso seja necessário, também é possível definir funções com nenhum ou vários argumentos, como no código abaixo:

In [10]:

```
def ola_idade(meu_nome,minha_idade):
    print(f'Olá {meu_nome}, você tem: {minha_idade} anos!')

ola_idade('Felipe',22)
```

```
Olá Felipe, você tem: 22 anos!
```

Assim como podem receber valores de entrada, as funções também podem produzir valores de saída, provenientes de determinadas operações.

Nos exemplos anteriores, apenas imprimimos um valor com a função print, sem retornar explicitamente um resultado. Já na Listagem 1, temos uma função que faz o cálculo do salário e retorna o valor a ser pago conforme o número de horas trabalhadas.

In [16]:

```
def media(nota1,nota2):
    media = (nota1 + nota2) / 2
    return media

    # set  ->  # get
minha_media = media(9,7)
print(minha_media)
```

```
8.0
```

Desafio 7.1 - Função para média onde o usuário informa os valores das notas

In [28]:

```
# Função Media():
    # Recebe duas notas -> Inputs
    # Calcula as duas notas
    # Retorna a média

def media():
    nota1 = float(input('Minha primeira nota: '))
    nota2 = float(input('Minha segunda nota: '))
    media = (nota1 + nota2) / 2
    return media

minha_media = media()
print("Minha média é:",minha_media)
```

```
Minha primeira nota: 10
Minha segunda nota: 8
Minha média é: 9.0
```

Desafio 7.2 - Função que chama outra função.

A primeira delas recebe duas notas do usuário e a segunda calcula a média

In [30]:

```
# Função ReceberNotas():
    # Recebe duas notas -> Inputs
    # Função Media(nota1,nota2)
        # Calcular as duas notas
        # Retornar a média

def media():
    nota1 = float(input('Minha primeira nota: '))
    nota2 = float(input('Minha segunda nota: '))
    return calcular_media(nota1,nota2)

def calcular_media(nota1,nota2):
    media = (nota1 + nota2) / 2
    return media

minha_media = media()
print("Minha média é:",minha_media)
```

```
Minha primeira nota: 10
Minha segunda nota: 8
Minha média é: 9.0
```

Desafio 7.3 - Função com funções dentro dela

A primeira função recebe duas notas e a segunda calcula a média.
A primeira então vai retornar essa média e esse retorno vai ser utilizado para uma terceira função saber se o aluno foi aprovado

In [32]:

```
# Função ReceberNotas():
    # Recebe duas notas -> Inputs
    # Função Media(nota1,nota2)
        # Calcular as duas notas
        # Retornar a média
    # Retorna essa média

# media = ReceberNotas()
# VerificarAprovacao(media)

def media():
    nota1 = float(input('Minha primeira nota: '))
    nota2 = float(input('Minha segunda nota: '))
    return calcular_media(nota1,nota2)

def calcular_media(nota1,nota2):
    media = (nota1 + nota2) / 2
    return media

def calcular_aprovacao(media):
    if(media >= 6):
        print('Aluno aprovado!')
    else:
        print('Aluno reprovado!')

calcular_aprovacao(media())
```

```
Minha primeira nota: 10
Minha segunda nota: 8
Aluno aprovado!
```

Aprendemos com os desafios:

- Uma função pode chamar outra função dentro dela
- Podemos retornar (return) uma função, contanto que essa função de retorno também retorne um valor
- Podemos enviar uma função por parâmetro, contanto que essa função enviada retorne um valor

Funções Nativas do Python

- Funções nativas do Python são usadas o tempo todo, já usávamos antes mesmo de saber que eram funções.
- Ex:
  - Print()
    - Mostra um valor para o usuário => Não retorna dado, então não possui return
  - List()
    - Cria uma lista => Retorna um dado, então é uma função com return
  - Int()
    - É uma função de cast (conversão de dado) que transforma o tipo do dado em inteiro
  - Str()
    - É uma função de cast (conversão de dado) que transforma o tipo da variavel para string
  - Float()
    - É uma função de cast (conversão de dado) que transforma o tipo da variavel para float
  - Bool()
    - É uma função de cast (conversão de dado) que transforma o tipo da variável para booleano
  - Type()
    - Mostra o tipo do dado
  - Dir()
    - Mostra as 'opções' de um determinado parâmetro (variável/classe/função)
  - Help()
    - Mostra uma ajuda para determinada função/classe/variável

In [8]:

```
print('Olá') # Função Nativa do Python
lista = list([1,2,3,4]) # Função Nativa que Retorna um Dado (lista)

# Funções de cast => Conversões de tipos de dado
meu_numero = int('123') # int('123')
minha_string = str(123) # str(123)
meu_numero_real = float('12.3') # float('12.3')

# type => Mostrar o tipo
type(meu_numero) # int
type(minha_string) # str
type(meu_numero_real) # float
```

```
Olá
```

Out[8]:

```
float
```

Funções nativas ficam no builtin do python.
Funções nativas podem ser usadas em qualquer lugar do nosso código, pois elas já vem com o Python.

In [10]:

```
# __builtin__ => "Já vem feito" => Funções/Variáveis nativas do Python
dir(__builtin__)
```

Out[10]:

```
['ArithmeticError',
 'AssertionError',
 'AttributeError',
 'BaseException',
 'BlockingIOError',
 'BrokenPipeError',
 'BufferError',
 'BytesWarning',
 'ChildProcessError',
 'ConnectionAbortedError',
 'ConnectionError',
 'ConnectionRefusedError',
 'ConnectionResetError',
 'DeprecationWarning',
 'EOFError',
 'Ellipsis',
 'EnvironmentError',
 'Exception',
 'False',
 'FileExistsError',
 'FileNotFoundError',
 'FloatingPointError',
 'FutureWarning',
 'GeneratorExit',
 'IOError',
 'ImportError',
 'ImportWarning',
 'IndentationError',
 'IndexError',
 'InterruptedError',
 'IsADirectoryError',
 'KeyError',
 'KeyboardInterrupt',
 'LookupError',
 'MemoryError',
 'ModuleNotFoundError',
 'NameError',
 'None',
 'NotADirectoryError',
 'NotImplemented',
 'NotImplementedError',
 'OSError',
 'OverflowError',
 'PendingDeprecationWarning',
 'PermissionError',
 'ProcessLookupError',
 'RecursionError',
 'ReferenceError',
 'ResourceWarning',
 'RuntimeError',
 'RuntimeWarning',
 'StopAsyncIteration',
 'StopIteration',
 'SyntaxError',
 'SyntaxWarning',
 'SystemError',
 'SystemExit',
 'TabError',
 'TimeoutError',
 'True',
 'TypeError',
 'UnboundLocalError',
 'UnicodeDecodeError',
 'UnicodeEncodeError',
 'UnicodeError',
 'UnicodeTranslateError',
 'UnicodeWarning',
 'UserWarning',
 'ValueError',
 'Warning',
 'WindowsError',
 'ZeroDivisionError',
 '__IPYTHON__',
 '__build_class__',
 '__debug__',
 '__doc__',
 '__import__',
 '__loader__',
 '__name__',
 '__package__',
 '__spec__',
 'abs',
 'all',
 'any',
 'ascii',
 'bin',
 'bool',
 'breakpoint',
 'bytearray',
 'bytes',
 'callable',
 'chr',
 'classmethod',
 'compile',
 'complex',
 'copyright',
 'credits',
 'delattr',
 'dict',
 'dir',
 'display',
 'divmod',
 'enumerate',
 'eval',
 'exec',
 'filter',
 'float',
 'format',
 'frozenset',
 'get_ipython',
 'getattr',
 'globals',
 'hasattr',
 'hash',
 'help',
 'hex',
 'id',
 'input',
 'int',
 'isinstance',
 'issubclass',
 'iter',
 'len',
 'license',
 'list',
 'locals',
 'map',
 'max',
 'memoryview',
 'min',
 'next',
 'object',
 'oct',
 'open',
 'ord',
 'pow',
 'print',
 'property',
 'range',
 'repr',
 'reversed',
 'round',
 'set',
 'setattr',
 'slice',
 'sorted',
 'staticmethod',
 'str',
 'sum',
 'super',
 'tuple',
 'type',
 'vars',
 'zip']
```

In [11]:

```
dir(list())
```

Out[11]:

```
['__add__',
 '__class__',
 '__contains__',
 '__delattr__',
 '__delitem__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getitem__',
 '__gt__',
 '__hash__',
 '__iadd__',
 '__imul__',
 '__init__',
 '__init_subclass__',
 '__iter__',
 '__le__',
 '__len__',
 '__lt__',
 '__mul__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__reversed__',
 '__rmul__',
 '__setattr__',
 '__setitem__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 'append',
 'clear',
 'copy',
 'count',
 'extend',
 'index',
 'insert',
 'pop',
 'remove',
 'reverse',
 'sort']
```

In [15]:

```
lista = [1,2,3,4]
lista.reverse # Chute 1
lista.reverse() # Chute 2
lista # Aprendi no chute que o reverse deixa a lista de trás pra frente (inverso)
```

Out[15]:

```
[4, 3, 2, 1]
```

Para mostrar uma ajuda de uma determinada função/classe/variavel eu posso usar a função help()

In [16]:

```
help(list().reverse)
```

```
Help on built-in function reverse:

reverse() method of builtins.list instance
    Reverse *IN PLACE*.
```

In [17]:

```
help(abs) # Retorna o valor absoluto
```

```
Help on built-in function abs in module builtins:

abs(x, /)
    Return the absolute value of the argument.
```

In [20]:

```
help(list)
```

```
Help on class list in module builtins:

class list(object)
 |  list(iterable=(), /)
 |  
 |  Built-in mutable sequence.
 |  
 |  If no argument is given, the constructor creates a new empty list.
 |  The argument must be an iterable if specified.
 |  
 |  Methods defined here:
 |  
 |  __add__(self, value, /)
 |      Return self+value.
 |  
 |  __contains__(self, key, /)
 |      Return key in self.
 |  
 |  __delitem__(self, key, /)
 |      Delete self[key].
 |  
 |  __eq__(self, value, /)
 |      Return self==value.
 |  
 |  __ge__(self, value, /)
 |      Return self>=value.
 |  
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |  
 |  __getitem__(...)
 |      x.__getitem__(y) <==> x[y]
 |  
 |  __gt__(self, value, /)
 |      Return self>value.
 |  
 |  __iadd__(self, value, /)
 |      Implement self+=value.
 |  
 |  __imul__(self, value, /)
 |      Implement self*=value.
 |  
 |  __init__(self, /, *args, **kwargs)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |  
 |  __iter__(self, /)
 |      Implement iter(self).
 |  
 |  __le__(self, value, /)
 |      Return self<=value.
 |  
 |  __len__(self, /)
 |      Return len(self).
 |  
 |  __lt__(self, value, /)
 |      Return self<value.
 |  
 |  __mul__(self, value, /)
 |      Return self*value.
 |  
 |  __ne__(self, value, /)
 |      Return self!=value.
 |  
 |  __repr__(self, /)
 |      Return repr(self).
 |  
 |  __reversed__(self, /)
 |      Return a reverse iterator over the list.
 |  
 |  __rmul__(self, value, /)
 |      Return value*self.
 |  
 |  __setitem__(self, key, value, /)
 |      Set self[key] to value.
 |  
 |  __sizeof__(self, /)
 |      Return the size of the list in memory, in bytes.
 |  
 |  append(self, object, /)
 |      Append object to the end of the list.
 |  
 |  clear(self, /)
 |      Remove all items from list.
 |  
 |  copy(self, /)
 |      Return a shallow copy of the list.
 |  
 |  count(self, value, /)
 |      Return number of occurrences of value.
 |  
 |  extend(self, iterable, /)
 |      Extend list by appending elements from the iterable.
 |  
 |  index(self, value, start=0, stop=9223372036854775807, /)
 |      Return first index of value.
 |      
 |      Raises ValueError if the value is not present.
 |  
 |  insert(self, index, object, /)
 |      Insert object before index.
 |  
 |  pop(self, index=-1, /)
 |      Remove and return item at index (default last).
 |      
 |      Raises IndexError if list is empty or index is out of range.
 |  
 |  remove(self, value, /)
 |      Remove first occurrence of value.
 |      
 |      Raises ValueError if the value is not present.
 |  
 |  reverse(self, /)
 |      Reverse *IN PLACE*.
 |  
 |  sort(self, /, *, key=None, reverse=False)
 |      Stable sort *IN PLACE*.
 |  
 |  ----------------------------------------------------------------------
 |  Static methods defined here:
 |  
 |  __new__(*args, **kwargs) from builtins.type
 |      Create and return a new object.  See help(type) for accurate signature.
 |  
 |  ----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |  
 |  __hash__ = None
```

O help também funciona enviando uma string por parâmetro.
Ele irá procurar uma função com aquele nome e retornar uma ajuda.

In [21]:

```
help('print')
```

```
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)
    
    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

Com a função help() aprendemos que:

- O parâmetro "sep" altera o separador em um print (,) => Padrão = espaço (" ")
- O parâmetro "end" altera o final da linha em um print => Padrão = quebra de linha ("\n")

In [23]:

```
print("Olá","mundo","eu","sou","o","Felipe", sep='-')
```

```
Olá-mundo-eu-sou-o-Felipe
```

In [25]:

```
print("Olá",end=' ')
print("Mundo",end=' ')
print("Eu",end=' ')
print("Sou",end=' ')
print("O",end=' ')
print("Felipe")
```

```
Olá Mundo Eu Sou O Felipe
```

In [ ]: