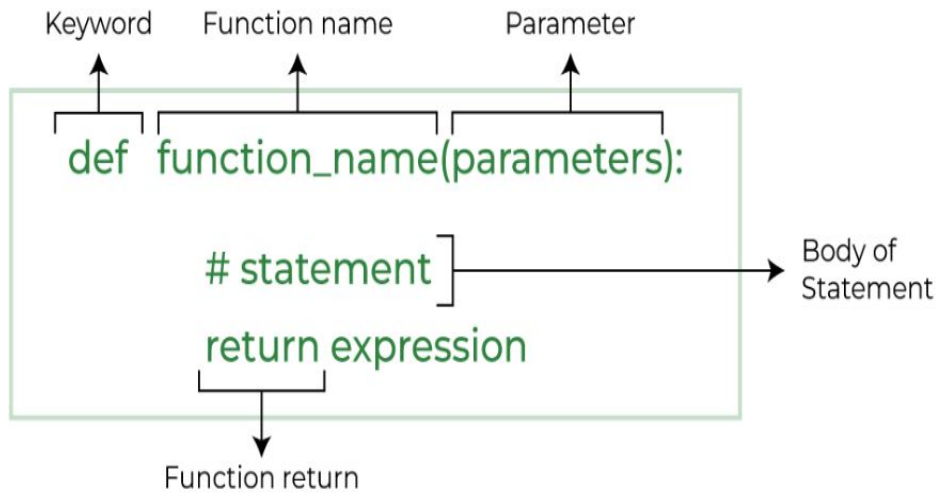


Laboratório de Programação

Armando Soares Sousa

Funções em Python

Uma função em uma LP corresponde a um bloco de instruções que, ao ser executado, pode retornar um resultado específico.



```
def f(x):  
    return (x + 1)  
  
x = 1  
# Chamando a função f(x)  
print(f"Para x = {x} o resultado da função f(x) é {f(x)}")  
x = 2  
# Chamando a função f(x)  
print(f"Para x = {x} o resultado da função f(x) é {f(x)}")  
x = 3  
# Chamando a função f(x)  
print(f"Para x = {x} o resultado da função f(x) é {f(x)}")
```

```
Para x = 1 o resultado da função f(x) é 2  
Para x = 2 o resultado da função f(x) é 3  
Para x = 3 o resultado da função f(x) é 4
```

Argumentos (parâmetros) de uma função

Em Python podemos ter os seguintes cenários para parâmetros de função:

1. Nenhum parâmetro
2. Apenas um parâmetro
3. N parâmetros

```
def bem_vindo():  
    print("Bem vindo as funções do Python!")  
  
def f(x):  
    return (x + 1)  
  
def g(x, w):  
    return (x + w)  
  
# Chamando a função bem_vind()  
bem_vindo()  
  
x = 1  
# Chamando a função f(x)  
print(f"Para x = {x} o resultado da função f(x) é {f(x)}")  
  
x, w = (2, 3)  
# Chamando a função g(x,w)  
print(f"Para x = {x} o resultado da função g(x,w) é {g(x,w)}")
```

```
Bem vindo as funções do Python!  
Para x = 1 o resultado da função f(x) é 2  
Para x = 2 o resultado da função g(x,w) é 5
```

Argumentos (parâmetros) de uma função

É possível definir valores "default" nas funções, ou seja, valores padrões podem ser passados para um, ou vários argumentos de uma função.

Também é possível chamar os parâmetros na ordem diferente da definição da função, caso na chamada da função sejam usados os nomes do parâmetro com a atribuição dos seus valores (chamado de *keywords arguments*).

```
def bem_vindo(usuario="Armando"):
    print(f"Olá {usuario}! Seja bem vindo as funções do Python!")

def f(x):
    return (x + 1)

def g(x, w):
    return (x + w)

# Chamando a função bem_vind()
bem_vindo()

x = 1
# Chamando a função f(x)
print(f"Para x = {x} o resultado da função f(x) é {f(x)}")

# Chamando a função g(x,w)
# usando valores explicitos de cada parâmetro
resultado_g_x_w = g(w=2, x=3)
print(f"Para x = 3 e w = 2 o resultado da função g(x,w) é {resultado_g_x_w}")
```

```
Olá Armando! Seja bem vindo as funções do Python!
Para x = 1 o resultado da função f(x) é 2
Para x = 3 e w = 2 o resultado da função g(x,w) é 5
```

Argumentos (parâmetros) de uma função

No Python, é possível passar um número variável de argumentos na função:

1. Via a declaração **args*. Neste caso, o *args* representa uma tupla de *n* valores (valor1, ..., valorN)
2. Via a declaração ***kwargs*. Neste caso, o *kwargs* representa dicionário de chaves e valores {chave1=valor1, chave2=valor2,... chaveN=valorN}

```
def minhas_palavras(*argv):  
    for arg in argv:  
        print(arg)
```

```
minhas_palavras('Hello', 'Welcome', 'to', 'Python Functions')
```

```
Hello  
Welcome  
to  
Python Functions
```

Argumentos (parâmetros) de uma função

No Python, é possível passar um número variável de argumentos na função:

1. Via a declaração **args*. Neste caso, o *args* representa uma tupla de *n* valores (valor1, ..., valorN)
2. Via a declaração ***kwargs*. Neste caso, o *kwargs* representa dicionário de chaves e valores {chave1=valor1, chave2=valor2,... chaveN=valorN}

```
def my_name(**kwargs):
    for key, value in kwargs.items():
        print(f"{{key, value}}")

def my_contacts(**kwargs):
    for key, value in kwargs.items():
        print(f"{{key, value}}")

# Meu nome:
print("Meu nome")
my_name(first='Armando', mid='Soares', last='Sousa')
print("")
# Meu contato
print("Meu contato")
my_contacts(nome='Maria Joaquina', email='mj@gmail.com', telefones=['988776655', '977886756'])
```

```
Meu nome
('first', 'Armando')
('mid', 'Soares')
('last', 'Sousa')

Meu contato
('nome', 'Maria Joaquina')
('email', 'mj@gmail.com')
('telefones', ['988776655', '977886756'])
```

Escopo de variáveis em Python

Um escopo em qualquer linguagem de programação é uma região do programa onde uma variável definida pode existir e além dessa variável ela não pode ser acessada.

Existem três locais onde as variáveis podem ser declaradas na linguagem de programação Python:

- Dentro de uma função ou bloco que é chamado de variáveis locais.
- Fora de todas as funções que são chamadas de variáveis globais.
- Na definição dos parâmetros da função que são chamados de parâmetros formais.

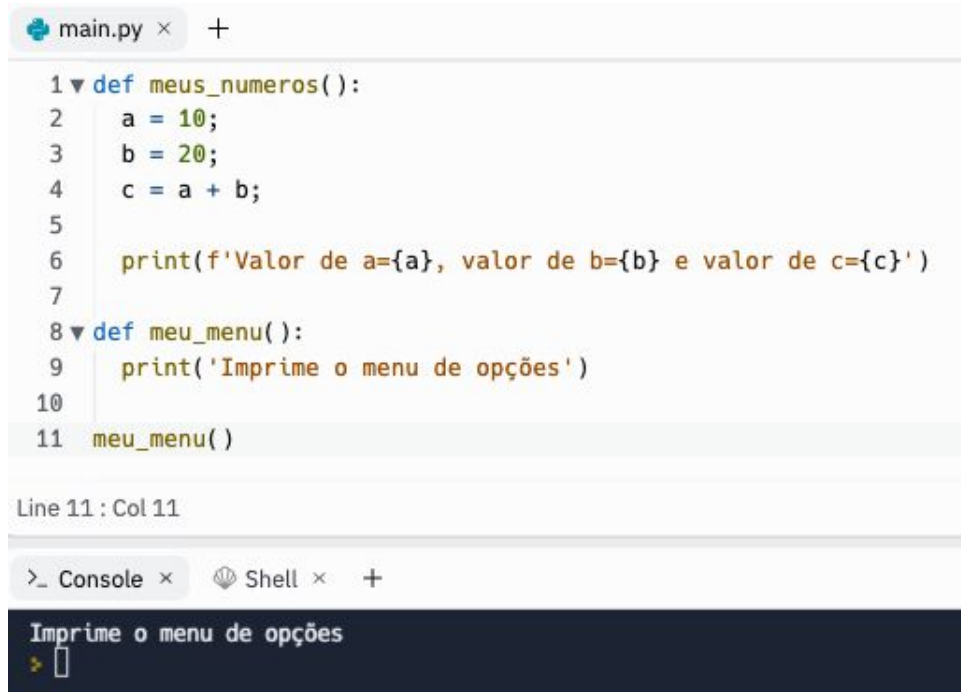
Variáveis locais

As variáveis que são declaradas dentro de uma função ou bloco são chamadas de **variáveis locais**.

Eles podem ser usados apenas por instruções que estão dentro dessa função ou bloco de código.

Variáveis locais não são conhecidas por funções fora delas.

No exemplo, todas as variáveis `a`, `b` e `c` são locais para a função `meus_numeros()`.



```
main.py x +  
1 ▼ def meus_numeros():  
2     a = 10;  
3     b = 20;  
4     c = a + b;  
5  
6     print(f'Valor de a={a}, valor de b={b} e valor de c={c}')
```



```
8 ▼ def meu_menu():  
9     print('Imprime o menu de opções')
```



```
10  
11 meu_menu()
```


Line 11 : Col 11

>_ Console x Shell x +

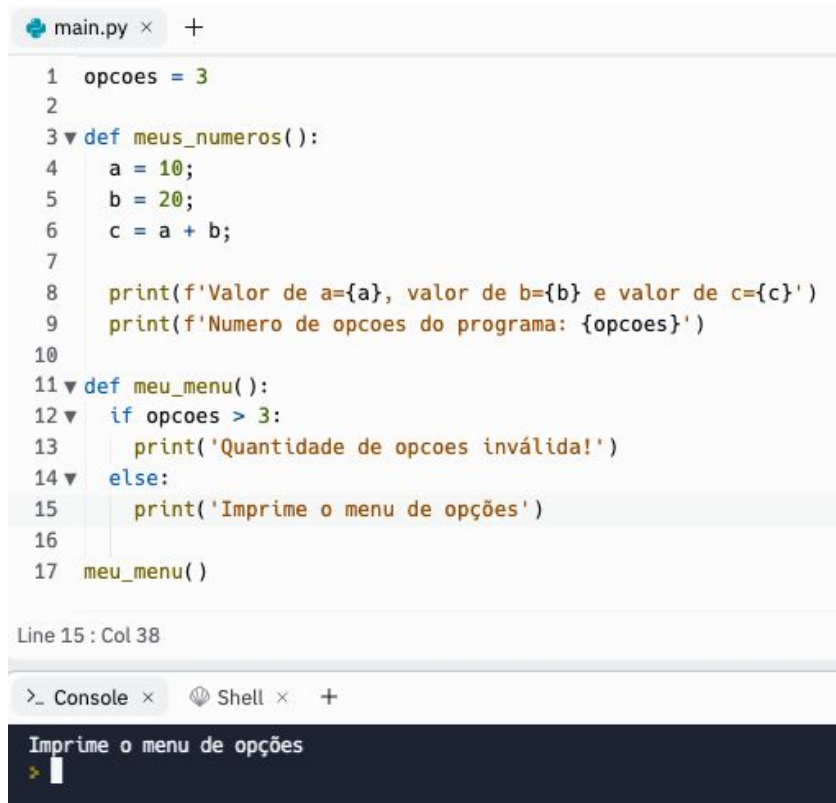
```
Imprime o menu de opções  
➤
```


Variáveis globais

As variáveis globais são definidas fora de uma função, geralmente no topo do programa.

As variáveis globais mantêm seus valores durante todo o tempo de vida do seu programa e podem ser acessadas dentro de qualquer uma das funções definidas para o programa.

Uma variável global pode ser acessada por qualquer função. Ou seja, uma variável global está disponível para uso em todo o seu programa após sua declaração.



```
main.py x +  
1 opcoes = 3  
2  
3 ▼ def meus_numeros():  
4     a = 10;  
5     b = 20;  
6     c = a + b;  
7  
8     print(f'Valor de a={a}, valor de b={b} e valor de c={c}')  
9     print(f'Numero de opcoes do programa: {opcoes}')  
10  
11 ▼ def meu_menu():  
12 ▼     if opcoes > 3:  
13         print('Quantidade de opcoes inválida!')  
14 ▼     else:  
15         print('Imprime o menu de opções')  
16  
17     meu_menu()  
  
Line 15 : Col 38  
  
_ Console x Shell x +  
Imprime o menu de opções  
➤
```

Números randômicos

Existe o módulo random que já fornece várias funções de randomização numérica.

import random

As mais importantes são:

random() → float: random value in [0.0, 1.0[

randint(a,b) → int: random value in [a, b]

choice(seq) → value: random item from seq sequence

shuffle(list) → items of list randomly reordered

```
import random

randomico = random.random()

numero_randomico_inteiro = random.randint(1,10)

escolha_da_lista = random.choice([2,4,6,8])

lista = [2,4,6,8]
random.shuffle(lista)

print(randomico)
print(numero_randomico_inteiro)
print(escolha_da_lista)
print(lista)
```

```
0.2831096277695153
5
6
[6, 4, 2, 8]
```

Básico de Data e tempo

Existe o módulo [datetime](#) que fornece várias funções relacionadas a data e tempo.

`import datetime`

Data atual

`today()`

Data atual completa

`now()`

Para formatar uma data

[strftime](#)() espera um padrão de string explicando como você deseja formatar sua data.

```
import datetime

# Mostra o dia atual do sistema
hoje = datetime.date.today()
print(hoje)

print("")
# Mostra a data atual completa do sistema
data_atual = datetime.datetime.now()
print(data_atual)

print(hoje.strftime('We are the %d, %b %Y'))
print(hoje.strftime('We are the %d/%m/%Y'))
print("")

print (data_atual.strftime("%Y-%m-%d %H:%M:%S"))
```

2022-12-06

2022-12-06 16:19:05.765671

We are the 06, Dec 2022

We are the 06/12/2022

2022-12-06 16:19:05