Definindo Funções

São pequenos trechos de códigos que realizam uma determinada tarefa e pode ser reutilizados, quando inseridas dentro de módulos.

Desde o início dos nossos estudos na Linaguagem Python, já tivemos contato com funções primárias, chamsda de **builtins**.

Exemplos: **print( )**, **input( )**, **int( )**, **max( )** e etc.

Definindo a primeira função

Em Python, a forma geral de definir uma função é:

```
def nome_da_funcao():
    bloco de instruções da função
```

```
def print_hi():
    print("Hi")
```

Observações

1. Perceba que dentro das nossas funções podemos utilizar outras funções;
2. Perceba que que nossa função só irá executar uma única tarefa;
3. Nossa função não recebe nenhum parâmetro de entrada;
4. Perceba que esta função não **retorna nada**.

Utilizando a sua primeira função

Você deverá primeiramente definir sua função como foi feito nos exemplos acima. Em seguida você deverá invocar sua função.

Para isso basta chamar o nome da sua função no seu código seguido de parênteses **()**.

Observe o exemplo abaixo:

```
def escreva():
    print("Parabéns !!!")

def cantar_parabens():
    print("Parabéns para você !")
```

```
    print("Nesta data querida !")
    print("Muitas felicidades !")
    print("Muitos anos de vida !")

escreva()
cantar_parabens()
print("*"*25)

canta = cantar_parabens
canta()
```

- No Python é possível **criar variáveis do tipo de uma função**.

Funções com retorno

Em Python quando uma função retona valores o retono é **None**.

Para retornar um valor no Python utilizamos a pavra reservada **return**.

Não é preciso necessariamente criar uma variável para receber o retorno de uma função. para isso podemos utilizar e chamar a operação diretamente.

Exemplo de uma função que retona um valor:

```
def dobro_numero_22():
    return 22*2

#Exemplo 1 - Chamando diretamente
print(dobro_numero_22())

#Exemplo 2 - Chamando diretamente
print(f"{dobro_numero_22()}")

#Exemplo 3 - Chamando o retorno por meio de variáveis
dobro = dobro_numero_22()
print(f"{dobro}")
```

Sobre a palavra **return**:

1. A palavra reservada **return** finaliza a função. Ou seja, ela sai da execução.
2. Em uma função pode existir diferentes **return**.
3. Podemos, em uma função, retornar qualquer tipo de dado e até mesmo **múltiplos valores**.

```
# Funções com vários **return**

def exemplo():
    var = True
```

```
    if var:
        return 3
    elif var is None:
        return 5
    else:
        return 7
```

- Vamos agora ver um exemplo onde será retorna vários valores na função **return**:

```
def outro_exemplo():
    return 1,2,3,4,5

print(outro_exemplo())
print(type(outro_exemplo()))
```

- Observe que quando retornamos diferentes valores será **gerada uma tupla**.

Funções com parâmetros

Tratando-se função podemos perceber que temos diferentes maneiras de implementar uma função:

- Funções sem entradas (**sem parâmetros**):

```
def hello():
    print("Hello !")
```

- Funções sem saídas (**sem retorno**):

```
def hello():
    print("Hello !")
```

- Funções com entradas e sem saídas (**com parâmetros e sem retorno**):

```
def dobro(numero):
    print(numero*2)
```

- Funções com entras e com saídas (**com parâmetros e retorno**):

```
def dobro(numero):
    print(numero*2)
    return numero*2
```

Importante: se durante a invocação/execução da função for informado um número errado de parâmetros ou argumentos, teremos um TypeError.

Argumentos *versus* parâmetros

- Parâmetros são variáveis declaradas na definição de uma função.
- Argumentos são os dados passados durante a execução da função.

Argumentos nomeados

```
def full_name(nome, sobrenome):
    print(f"{nome} {sobrenome}")
    return f"{nome} {sobrenome}"

#Utilizando argumentos nomeados
full_name(nome="Marianny", sobrenome="Mariano")
full_name(sobrenome="Mariano", nome="Marianny")
```

Função com parâmetro padrão

Vamos ver como criar Funções onde a passagem de parâmetros seja opcional:

```
def expoente(numero, potencia):
    return numero**potencia

# Vamos refatorar a função _EXPOENTE_anterior para os parâmetros serem opcionais

def expoente2(numero, potencia=2):
    return numero**potencia

print(expoente(2,3))
print(expoente2(2))
print(expoente2(2,3))
```

Importante lembrar que os parâmetros com valores default DEVEM sempre estar ao final da declaração.

Parâmetros opcionais, permite:

- Funções mais flexíveis;

- Evitar erros com parâmetros incorretos
- Nos permite trabalhar com exemplos mais legíveis de códigos

---

Documentando funções com Docstrings

Entendendo o **args*

Entendendo o **kwargs*
