FUNÇÕES

Ruan Pablo Medeiros

FUNÇÕES - O QUE SÃO?

- Função está intimamente conectada a palavra rotina.
- Uma rotina é algo do nosso cotidiano que repetimos. Ex: Acordar, tomar café, tomar banho, escovar os dentes, vestir-se, ir para o serviço...
- Quando desejamos que um trecho do nosso código seja repetido em diferentes partes do nosso programa, podemos definir ele como uma função e o executar sempre que for necessário.

FUNÇÕES - EXEMPLOS DE FUNÇÕES DO PYTHON

- Desde os nossos primeiros programas em Python já utilizamos algumas funções predefinidas pela linguagem, por exemplo: print(), input(), len()...
- As funções acima já são definidas no próprio Python, porém podemos definir nossas próprias funções customizadas.

FUNÇÕES - CRIANDO FUNÇÕES

- Para criar nossas funções seguimos uma estrutura básica:

```
def nome(<parametro1>, <parametro2>, ...):
    <comandos>
    return
```

FUNÇÕES - CRIANDO FUNÇÕES

- O def no início de nossas definições de funções, é uma palavra reservada da linguagem que serve para informar ao programa que aquele trecho de código é uma função que estamos criando.
- Em seguida vem o nome da função, esse nome vai ser chamado dentro do nosso código sempre que quisermos executar a função.
- Depois do nome vem os parênteses() que todas as funções devem possuir em sua definição. É dentro dos parênteses onde definimos os parâmetros dessa função.

FUNÇÕES - CRIANDO FUNÇÕES

- Depois dos parênteses temos ":" que serve para iniciar o bloco da nossa função.
- Nosso bloco da função (Todos os comandos que a função vai executar) devem ser identados.
- E por fim nossa função pode possuir ou não um retorno, utilizaremos a palavra reservada return.

FUNÇÕES - CRIANDO FUNÇÕES

- Nossa primeira função:

```
1 def ola_mundo():
2     print("Olá mundo!")
3
4 ola_mundo()
```

FUNÇÕES - CRIANDO FUNÇÕES

- Note que quando a função ola_mundo é definida ela não executa o código que está dentro de seu bloco, ela só executa quando é chamada em alguma parte do programa, como podemos ver na linha 4 do slide anterior a chamada ola_mundo().

CRIANDO FUNÇÕES - PARÂMETROS

- Podemos também criar funções com parâmetros.

```
1 def soma(x, y):
2     soma = x + y
3     print(f"A soma de {x} + {y} é {soma}")
4
5 soma(3, 4)
```

CRIANDO FUNÇÕES - PARÂMETROS

- Veja que a nossa função soma do slide anterior foi criada com 2 parâmetros, sendo eles "x" e "y".
- Ao chamar a função soma(linha 5), precisamos passar como argumento os valores que nossos parâmetros vão receber para que a função execute.

CRIANDO FUNÇÕES - PARÂMETROS VS ARGUMENTOS

- Mas afinal, o que são parâmetros e argumentos?
- Parâmetros: podemos entender os parâmetros das funções como variáveis que vão ser utilizadas dentro do nosso código da função.
- Argumentos: são os valores passados na chamada da função para seus respectivos parâmetros, mesma ideia de atribuir um valor a uma variável.
- Em resumo, na linha 1 onde definimos soma, os parâmetros da função são "x" e "y", ao chamar a função soma na linha 5 passamos como argumentos 3 e 4, logo dentro do bloco para essa execução x = 3 e y = 4.

CRIANDO FUNÇÕES - PARÂMETROS VS ARGUMENTOS

- Note que se fizermos outra chamada da função soma na linha 6, com os argumentos 7 e 8, nós teremos outro resultado para a execução da função.

```
1 def soma(x, y):
2     soma = x + y
3     print(f"A soma de {x} + {y} é {soma}")
4
5 soma(3, 4)
6 soma(7,8)
```

ARGUMENTO POSICIONADO VS NOMEADO

- Temos duas formas de passar nossos argumentos para as funções:
- 1 – Posicionado: a ordem dos argumentos importa, no caso da função abaixo o primeiro argumento é associado ao parâmetro x e o segundo ao y.

```
1 def soma(x, y):
2     soma = x + y
3     print(f"A soma de {x} + {y} é {soma}")
4
5 soma(3, 4)
```

ARGUMENTO POSICIONADO VS NOMEADO

- Temos duas formas de passar nossos argumentos para as funções:
- 2– Nomeado: a ordem dos parâmetros não importa, desde que tenha sido nomeado o argumento na chamada da função.

```
1 def soma(x, y):
2     soma = x + y
3     print(f"A soma de {x} + {y} é {soma}")
4
5 soma(y=4, x=3)
```

ARGUMENTO POSICIONADO VS NOMEADO

- Existe uma particularidade para argumentos nomeados, sempre que utilizar uma mistura de argumentos nomeados com posicionais, a partir do momento que utilizar um nomeado o restante dos argumentos também deve ser nomeado.

```
1 def soma(x, y, z):
2     soma = x + y + z
3     print(f"A soma de {x} + {y} + {z} é {soma}")
4
5 soma(3, z=4, 2) #ERRADO
```

ARGUMENTO POSICIONADO VS NOMEADO

- Existe uma particularidade para argumentos nomeados, sempre que utilizar uma mistura de argumentos nomeados com posicionais, a partir do momento que utilizar um nomeado o restante dos argumentos também deve ser nomeado.

```
1 def soma(x, y, z):
2     soma = x + y + z
3     print(f"A soma de {x} + {y} + {z} é {soma}")
4
5 soma(3, z=4, y=2) #CERTO
```

VALORES PADRÕES

- Quando não queremos passar determinado argumento para uma função, podemos definir um valor padrão para o seu parâmetro.

```
1 def soma(x, y, z=0):
2     soma = x + y + z
3     print(f"A soma de {x} + {y} + {z} é {soma}")
4
5 soma(3, 4)
7 soma(3, 4)
8 soma(3, 4, 5)
```

VALORES PADRÕES

- Mas e se por um acaso quisermos saber se foi passado o argumento para um parâmetro que possui valor padrão?

```
1 def soma(x, y, z=None):
2     if z is None:
3         print(f"A soma de {x} + {y} é {soma}")
4     else:
5         print(f"A soma de {x} + {y} + {z} é {soma}")
6
7 soma(3, 4)
8 soma(3, 4)
9 soma(3, 4, 5)
```

VALORES PADRÕES

- Existe uma particularidade no valor padrão que é parecida com a do argumento nomeado. A partir do momento que se define um valor padrão para um parâmetro, todos os parâmetros seguintes precisam possuir um valor padrão.

```
1 def soma(x, y=2, z): #ERRADO
2     soma = x + y + z
3     print(f"A soma de {x} + {y} + {z} é {soma}")
4
5 soma(3, 4, 2)
```

VALORES PADRÕES

- Existe uma particularidade no valor padrão que é parecida com a do argumento nomeado. A partir do momento que se define um valor padrão para um parâmetro, todos os parâmetros seguintes precisam possuir um valor padrão.

```
1 def soma(x, y=2, z=0): #CERTO
2     soma = x + y + z
3     print(f"A soma de {x} + {y} + {z} é {soma}")
4
5 soma(3, 4, 2)
```

ESCOPO DE FUNÇÕES

- Escopo significa local onde aquele código pode atingir
- Existe o escopo global e o local
- Escopo global é o escopo onde todo código é alcançável.
- Escopo local é o escopo onde apenas nomes do mesmo local podem ser alcançados.
- Não temos acesso a nomes de escopos internos em escopos externos
- A palavra global faz uma variável de escopo externo ser a mesma do escopo interno.

RETORNO DE FUNÇÕES

- Ao fazer uma chamada para uma função, como resultado podemos ter um valor que substituirá a chamada da função pelo valor retornado dessa função.
- Para retornar um valor de dentro de uma função utilizamos a palavra return.
- Imagine uma função como sendo um liquidificador, onde passamos os ingredientes(parâmetros/argumentos) e ao misturar nos gera um resultado(retorno da função)

RETORNO DE FUNÇÕES

- Uma função pode ou não explicitar um retorno.
- Quando não explicitamos o retorno de uma função ele é None.
- Podemos armazenar o retorno de uma função dentro de uma variável para ser utilizado posteriormente dentro do nosso código.

RETORNO DE FUNÇÕES

- Função com retorno:

```
1 def soma(x,y):
2     return x + y
3
4 soma1 = soma(1,2) # O valor de 1 + 2 é retornado da função e armazenado em soma1
5 print(soma1) # Saída: 3
```

RETORNO DE FUNÇÕES

- Sempre que uma função alcançar o return ela retorna para o trecho do código que a chamou e tudo que vem depois do return dentro da função não é executado:

```
1 def soma(x,y):
2     return x + y
4     print(x+y) # Trecho de código inalcançável
3
4 soma1 = soma(1,2) # O valor de 1 + 2 é retornado da função e armazenado em soma1
5 print(soma1) # Saída: 3
```

RETORNO DE FUNÇÕES

- Sempre que uma função alcançar o return ela retorna para o trecho do código que a chamou e tudo que vem depois do return dentro da função não é executado:

```
1 def eh_par(numero):
2     if numero % 2 == 0:
3         return True
4     return False # Trecho de código alcançável somente se o return dentro no if não for alcançado
5
6 print(eh_par(2)) # Saída: True
```

EXERCÍCIOS

1. Escreva uma função chamada saudacao, essa função tem como parâmetro uma string que é o nome da pessoa. Ao ser executada essa função deve mostrar na tela: "Olá, {nomeDaPessoa}!", onde nomeDaPessoa é o valor do parâmetro recebido como argumento na chamada da função.
2. Escreva uma função que representa o menu de uma aplicação de banco, a função deve mostrar em tela o menu abaixo e retornar a opção que o usuário informou:

```
---------------------------
   Menu
---------------------------
Escolha uma opcção:
(a) consulta saldo
(b) saque e
(c) depósito e
(d) sair.
>
```

EXERCÍCIOS

3. Explorando a Maioridade:

Crie uma função verifica_maioridade(idade) que receba a idade de uma pessoa como parâmetro. A função deve retornar True se a pessoa for maior de idade (no Brasil, 18 anos ou mais) e False caso contrário.

4. Calculando o IMC:

Desenvolva uma função calcula_imc(peso, altura) que receba o peso e a altura de uma pessoa como parâmetros. A função deve calcular o IMC (Índice de Massa Corporal) dessa pessoa e retornar o valor. Utilize a fórmula: IMC = peso / (altura * altura).

EXERCÍCIOS

5. Analisando Triângulos:

- Crie uma função classifica_triangulo(lado1, lado2, lado3) que receba os comprimentos dos lados de um triângulo como parâmetros. A função deve retornar uma string que classifique o triângulo de acordo com seus lados:
  - Equilátero: Se todos os lados forem iguais.
  - Isósceles: Se dois lados forem iguais.
  - Escaleno: Se todos os lados forem diferentes.
  - Não é triângulo: Se a soma de quaisquer dois lados for menor que o terceiro.

EXERCÍCIOS

6. Brincando com Números:

- Desenvolva uma função gera_lista_aleatoria(n, limite_inferior, limite_superior) que receba como parâmetros a quantidade de números (n), o limite inferior (limite_inferior) e o limite superior (limite_superior). A função deve gerar uma lista com n números aleatórios entre o limite_inferior e o limite_superior (inclusive).
- A lista gerada deve ser retornada.