

Curso Preparatório

Semana 7:
Funções

Admissão 2025





Funções em Python

- Em Python, uma **função** é uma sequência de comandos que executa alguma tarefa e que tem um nome. A sua principal finalidade é nos ajudar a organizar programas em pedaços que correspondam a como imaginamos uma solução do problema.
 - São rotinas criadas para executar um determinado comando.



Funções em Python

- Até agora, já usamos muitas funções que já vem na linguagem Python:
 - `print()`
 - `len()`
 - `int()`
 - `input()`
 -
- Uma função pode já vir na linguagem, ou você pode criar uma do zero.



Funções em Python

- As funções servem para guardarmos um bloco de código que poderá ser reutilizado quando precisarmos. É comum nos códigos estruturados fazer uso de funções.
- As funções também servem para organizar melhor o código fonte. Podemos ter funções que somente imprimirão em tela algum valor e teremos as funções que retornarão valores.
- Quando for usar funções, as vezes precisamos passar os dados a elas. Existem dois métodos de passagem de parâmetros geralmente usados:
 - por valor e por referência.
- Quando um parâmetro é passado por valor, uma cópia do valor contido é passada à função, e este valor inicia uma variável local. Quando um parâmetro é passado por referência, o endereço de memória em que a variável está é passado para a função. A função tem acesso real ao dado passado, e toda a alteração que ocorrer dentro da função irá impactar a variável.



Funções em Python

- Em Python usa a palavra reservada **def** para criar funções (defined function).
- A sintaxe de uma função é definida por três partes: nome, parâmetros e corpo, o qual agrupa uma sequência de linhas que representa algum comportamento. A sintaxe de uma definição de função é:

def *NOME*(**PARÂMETROS**):

____**COMANDOS**

NOME: nome que vai ser chamada a função, para usar essa função no código chame pelo *NOME*()

PARÂMETROS: A função pode ou não ter passagem de parâmetros.



Funções em Python

- No código abaixo, temos um exemplo de declaração de função em Python. Me diga o que essa função faz?

```
1 def linhas():  
2     print("="*20)
```

- Para “usar” essa função **linhas()**, basta você chamar no código em Python assim:

```
1 def linhas():  
2     print("="*20)  
3  
4     linhas()  
5     print("Jogo Par ou Impar")  
6     linhas()
```



Funções em Python

- Diferente de JAVA que a função podia vir antes ou depois da função **main()**, em Python, com não tem estrutura de código, a obrigação é que a definição da função seja feita antes dela ser usada (chamada). Veja os exemplos abaixo e diga qual deles você acha que esta errado?

```
1
2
3 def linhas():
4     print("="*20)
5     linhas()
6     print("Jogo Par ou Impar")
7     linhas()
8
```

```
1 def linhas():
2     print("="*20)
3
4
5
6
7
8     linhas()
9     print("Jogo Par ou Impar")
10    linhas()
```


```
1 linhas()
2 print("Jogo Par ou Impar")
3 linhas()
4
5 def linhas():
6     print("="*20)
7
```



Funções em Python

- Na passagem de parâmetros, não é preciso dizer o tipo de dado que a função vai receber, mas precisa identificar elas, o nome do parâmetro que você atribui na criação da variável, vai funcionar apenas para a função.
- Digamos que a função **linha()** anterior vai mostrar, além das linhas um nome, então poderíamos passar o seguinte parâmetro. Perceba a função, o uso e a saída na figura:

```
1 def linha(nome):
2     print("="*20)
3     print(nome)
4
5
6 txt = "Marcelo"
7 linha(txt) ↔ linha("Marcelo")
```



Nesse exemplo **nome** recebe o valor de **txt**



Funções em Python

- É possível passar mais de 1 parâmetro para a função, por exemplo, se você quiser pode passar números, textos, inclusive números e textos ao mesmo tempo.

- Me diga qual vai ser a saída do código ao lado:
- **nome** é string, **numero** é inteiro, então:

```
MarceloMarcelo
10
```

```
1 def soma(a,b):
2     print(a*2)
3     print(b*2)
4
5 numero = 5
6 nome = "Marcelo"
7 soma(nome, numero)
```

- O que acontece se na linha 7 tiver **soma(numero, nome)**?

```
10
MarceloMarcelo
```



Funções em Python

- Variáveis **Globais** e **Locais**, tenha atenção:
- Qual valor final de **x** no código abaixo?

```
1 def atencao(a):  
2     print("Dentro da função, a passa a valer x:",a)  
3     x = 5  
4     print("Dentro da função, x passa a valer:",x)  
5  
6 #programa principal  
7 x = 10  
8 print("Antes da função, x vale:",x)  
9 atencao(x)  
10 print("Depois da função, x vale:",x)
```

```
Antes da função, x vale: 10  
Dentro da função, a passa a valer x: 10  
Dentro da função, x passa a valer: 5  
Depois da função, x vale: 10
```

- Porque **x** na linha 3 é uma variável local de **atencao()**, enquanto **x** na linha 9 é variável global para o programa principal.
- Adicione na linha 3 do código anterior a expressão: **global x**, veja a saída.
 - Nesse caso, não cria a variável local **x**, vai considerar a variável apenas a global.



Funções em Python

- Uma das vantagens das funções de Python é que ela aceita como parâmetros, variáveis conjuntos, fazendo com que você não precise sempre colocar um número fixo de parâmetros:
- Para isso basta usar o * na criação da função antes do parâmetro, assim:

```
1 def funcao(*num):  
2     print(num)  
3  
4  
5 funcao("Lucas", "Ana")  
6 funcao(1, 5, 7, 4, 10)  
7 funcao(10)  
8 funcao("Lucas", 10, "Kelly", 15, 15.6)
```

Na saída, temos o que?

TUPLAS, inclusive podemos usar as funções prontas que já vem para tuplas, como por exemplo a `len()`.

```
✓ ↗ 📄  
( 'Lucas', 'Ana' )  
(1, 5, 7, 4, 10)  
(10,)  
( 'Lucas', 10, 'Kelly', 15, 15.6)
```



Funções em Python

- Além das Tuplas, as funções podem receber listas, que podem ser alteradas.
- No código abaixo, é passado uma lista como parâmetro de uma função chamada dobro(), e a função vai mostrar o dobro dos números da lista. A função vai mudar a lista para sempre, olhe no código à direita, o print antes (linha 7) e o print depois (linha 9) do uso da função na linha 8, qual vai ser saída?

```
1 def dobro(lista):
2     for i in range(0, len(lista)):
3         lista[i] *= 2
4     print(lista)
5
6
7 numeros = [1,3,5,7,9]
8 dobro(numeros)
```

```
1 def dobro(lista):
2     for i in range(0, len(lista)):
3         lista[i] *= 2
4
5
6 numeros = [1,3,5,7,9]
7 print("Antes da função:", numeros)
8 dobro(numeros)
9 print("depois da função:", numeros)
```



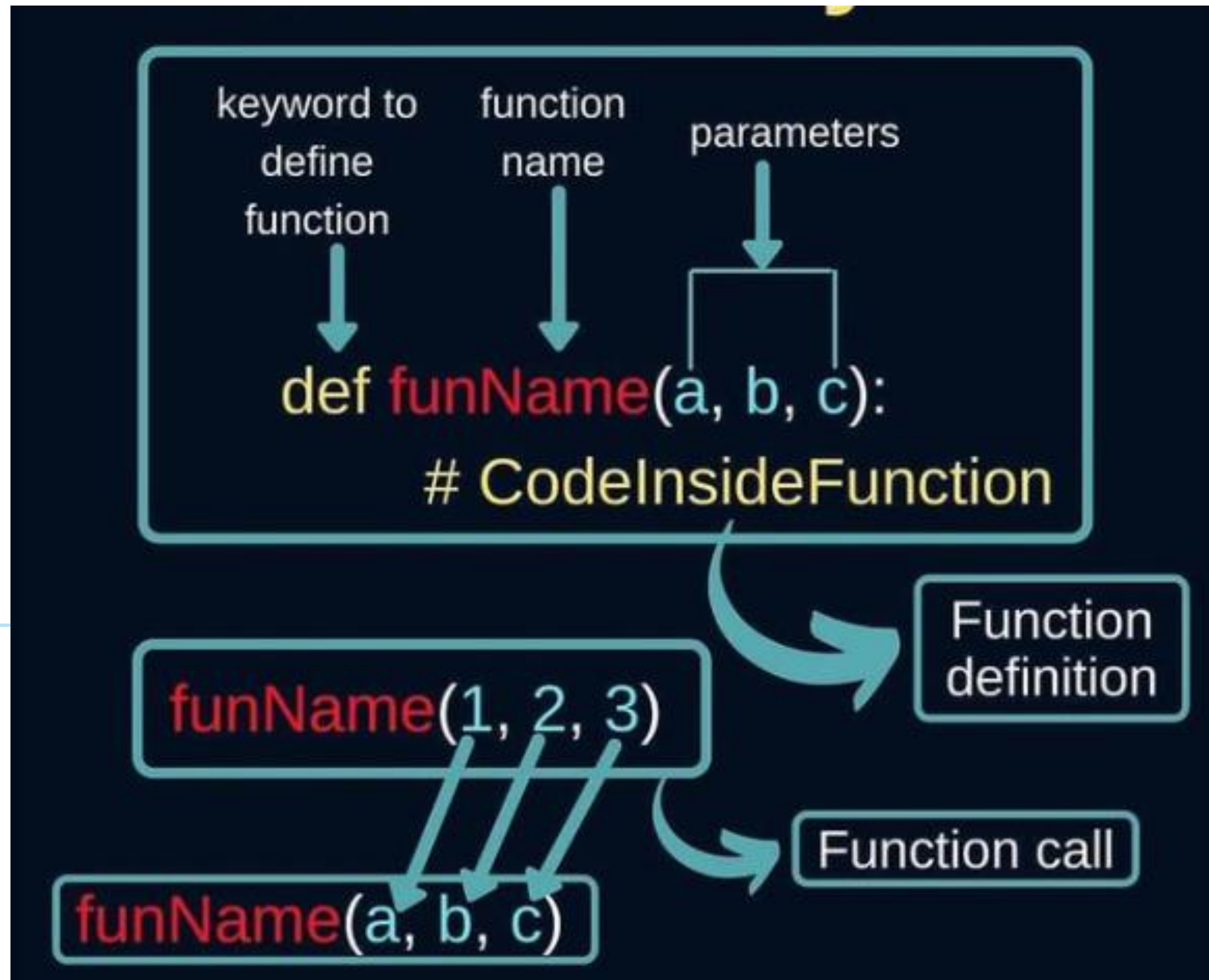
Prática - Funções

- Crie um programa que tenha 2 funções, uma chamada **menu()** para organizar uma impressão de dados e outra chamada **area()** para calcular e mostrar a área de um terreno retangular:

```
1 def menu(txt):  
2     print("-"*25)  
3     print(txt)  
4     print("-"*25)  
5  
6 def area(x,y):  
7     print("A área é:",x*y)  
8
```

Ao lado estão as funções, complete o código para ter uma saída próxima a essa abaixo:

```
-----  
Área de um terreno  
-----  
  
Qual largura: 15  
Qual altura: 8  
A área é: 120.0
```



@python.joy

<https://www.instagram.com/python.joy/>



Prática - Funções

Retorno de uma função:

- Muitas vezes você não quer que uma função imprima algo, mas que faça uma rotina e depois retorne algum valor, alguma coisa.
- Assim como você já viu em JAVA, Python também permite que as funções retornem um valor/informação:
- Para isso, continua usando a mesma palavra reservada, o **return**.
- As vezes queremos usar o valor de uma função no programa principal, outra vantagem é que podemos imprimir direto uma função que tem o **return()**.



Prática - Módulos

- Vamos fazer as funções de operações matemáticas em Python:
- Uma função **soma()**, uma **sub()** e outra **mult()**. Todas tem que retornar o valor.
- No programa, leia 2 números quaisquer e mostre os resultados das funções.

```
1 def soma(x,y):  
2     return x+y  
3  
4 def sub(x,y):  
5     return x-y  
6  
7 def mult(x,y):  
8     return x*y  
9  
10 num1 = float(input("Numero1: "))  
11 num2 = float(input("Numero2: "))  
12 print("Soma:", soma(num1,num2))  
13 print("Subtração:", sub(num1,num2))  
14 print("Multiplicação:", mult(num1,num2))
```




Prática - Módulos

- Crie um programa que tenha uma função chamada voto() que vai receber o ano de nascimento de uma pessoa. Essa função deve retornar um texto (não impressão), indicando se a pessoa “JÁ PODE VOTAR”, “NÃO PODE VOTAR”, “DEVE VOTAR”.
- **DICA:** Use a função `date.today()` para calcular a idade da pessoa a partir do nascimento.

```
1 from datetime import date
2 def voto(ano):
3     idade = date.today().year - ano
4     print("Você tem", idade, "anos")
5     if (idade < 16):
6         return "NÃO PODE VOTAR"
7     elif (idade >= 16 and idade <= 18):
8         return "JÁ PODE VOTAR"
9     else:
10        return "DEVE VOTAR"
11
12
13 while (True):
14     nascimento = int(input("Que ano você nasceu? "))
15     if (nascimento > 1000 and nascimento < 2021):
16         break
17 print(voto(nascimento))
```



Módulos

- Modularizar significa criar módulos, ou dividir algo grande em coisas menores. No caso de um código, é tornar o programa cada vez mais organizado e menor, melhorando a leitura e mudanças futuras. Resumindo é dividir um programa grande em pequenos pedaços (módulos).
- Infelizmente não é possível criar módulos com o onlineGDB, porque você precisa ter a estrutura de pastas, lembra como era com o HTML e CSS? É quase parecido:
 - Os arquivos `.py` precisam estar na mesma pasta;
 - É preciso importar um código no outro;
 - Você precisa chamar a função da seguinte maneira:
 - **import** *nome_arquivo.py*.funcao



Módulos

- Exemplo: Abaixo tem 3 arquivos .py (menu, calc e principal).
- **principal.py** importa (**import** nas linhas 1 e 2) os dois outros módulos
- **principal.py** usa **menu.py** na linha 4 e usa **calc.py** nas linhas 8, 9, 10 e 11.

```
menu.py X
menu.py > impress
1 def impress(mensagem):
2     print("="*25)
3     print(mensagem)
4     print("="*25)

calc.py X
calc.py > div
1 def soma(x,y):
2     return x+y
3
4 def sub(x,y):
5     return x-y
6
7 def mult(x,y):
8     return x*y
9
10 def div(x,y):
11     if (x>y):
12         return x/y
13     else:
14         print("Escolha um denominador maior")

principal.py X
principal.py > ...
1 import menu
2 import calc
3
4 menu.impress("Calculador")
5
6 num1 = int (input("Digite um numero: "))
7 num2 = int (input("Digite um numero: "))
8 print("Soma:",calc.soma(num1,num2))
9 print("Subtração:",calc.sub(num1,num2))
10 print("Multiplcação:",calc.mult(num1,num2))
11 print("Divisão:",calc.div(num1,num2))
```



Módulos

- Como já sabemos, Python já tem algumas funções implementadas, e que elas são chamadas de módulo. Você pode encontrar isso como sendo bibliotecas ou pacotes de Python:
- Alguns exemplos de funções úteis e que já usamos:
 - **math()** <-
 - math.sqrt() – Raiz quadrada
 - math.floor() – Arredonda para baixo
 - math.ceil() – Arredonda para cima
 - **date()**
 - date.today()
 - **pytube** <- Pode baixar vídeos do youtube
 - **pywhatkit** <- tem acesso ao whatsapp, para enviar mensagem, por exemplo.
 - pywhatkit.sendwhatmsg("+55numero", "mensagem", hora, minutos)
 - **turtle** <- Para criar desenhos
 - turtle.screen() – Abre uma janela para desenho



Módulos

- Se você quiser, ainda pode mudar o nome do método, usando o as, como no exemplo abaixo:
- `import pywhatkit as zap`
- `Import math as matematica`
- Abaixo alguns dicas de pacotes que possam ser úteis:
- <https://terminalroot.com.br/2019/12/as-30-melhores-bibliotecas-e-pacotes-python-para-iniciantes.html>
- <https://docs.python.org/pt-br/3/library/functions.html>
- <https://medium.com/reflex%C3%A3o-computacional/m%C3%B3dulo-turtle-d8949db55008>