



# INFINITY SCHOOL

VISUAL ART CREATIVE CENTER

AULA 04 – FUNÇÕES I

# O QUE IREMOS APRENDER

- 01** APRESENTAÇÃO
- 02** PARADIGMA FUNCIONAL
- 03** SINTAXE, DEFINIÇÃO E ESCOPO
- 04** PARÂMETROS, RETORNO E ARGUMENTOS
- 05** ROTINAS E UNIDADES LÓGICAS
- 06** BOAS PRÁTICAS
- 07** INTRODUÇÃO AULA 04

# Apresentação

## O que é uma função?

Uma função é como uma mini-fábrica que realiza uma tarefa específica.

Imagine ter uma máquina que faz suco. Você insere frutas (entrada), a máquina processa (corpo da função) e, no final, você obtém suco (saída).




# Paradigma Funcional

## O que é uma função?

O paradigma funcional é um estilo de programação onde a base são as funções. Ele foca em realizar computações através da avaliação de funções, evitando mudança de estado e dados mutáveis.

# Sintaxe, Definição e Escopo

**Sintaxe e Definição:** Aqui, `def` é a palavra-chave que define uma função, `saudacao` é o nome da função, e o que está dentro da função é o corpo.



```
1  #      nome da função
2  #      \ /
3  def    saudacao  ():
4
5      print('olá')
6
7  #execução
8  saudacao() # exibir olá no terminal
```

# Parâmetros, Retorno e Argumentos

## Parâmetros e Argumentos

O escopo de uma função é a região do código onde ela existe e pode ser utilizada.

## Retorno Padrão

Mesmo que uma função apenas imprima algo, ela tem um retorno padrão, que é `None`.

Isso ocorre porque em Python toda função deve retornar algo, mesmo que seja o valor `None`.

# Parâmetros, Retorno e Argumentos

## Parâmetros e Argumentos

Parâmetros são as variáveis que a função recebe e argumentos são os valores que você passa para os parâmetros.

```
1  #      nome da função
2  #      \/
3  #      \/      parametro
4  #      \/      \/
5  def      saudacao      (nome):
6
7      print(f'olá ${nome}')
8
9  #execução
10 #      argumento
11 #      \/
12 saudacao('augusto') # exibir olá agosto no terminal
13
14 sobrenome_nome = 'santana' # variavel que será passada como argumento
15
16 saudacao(sobrenome_nome) # exibir olá santana no terminal
```



# Parâmetros, Retorno e Argumentos

## Retorno

Toda função retorna algo por padrão, se por exemplo, você não adicionar o return, ela retornará `None` por baixo dos panos.

O `return` é usado para especificar o que a função deve retornar, sempre que usar o `return` e quiser exibir o resultado obtido, terá que ser usado o `print()` pós o return não exibi nada na tela

```
1  #      nome da função
2  #      \/
3  #      \/      parametro
4  #      \/      \/
5  def      saudacao      (nome):
6
7      return f'olá ${nome}'
8
9  #execução
10
11  sobrenome_nome = 'santana' # variavel que será passada como argumento
12
13  print(saudacao(sobrenome_nome)) # exibir olá santana no terminal
```



# Rotinas e Unidades Lógicas

## Rotinas

Rotinas realizam ações, mas não retornam valores.



```
1  #      nome da função
2  #      \ /
3  def    saudacao  ():
4
5      print('olá')
6
7  #execução
8  saudacao() # exibir olá no terminal
```

# Rotinas e Unidades Lógicas

## Unidades Lógicas

Resolvem problemas específicos e retornam valores.



```
1 def nome_completo(nome,sobrenome):  
2     return f'{nome} {sobrenome}'  
3  
4 idade=21  
5 #           função      argumento1      argumento2  
6 nome_idade = f'meu nome {nome_completo('augusto', 'santanta')} e idade é {idade}'  
7  
8 print(nome_idade)
```

# Boas Práticas

## Documentação

Documentar suas funções é essencial para que outros desenvolvedores (ou você mesmo no futuro) possam entender o que a função faz, quais parâmetros ela recebe e o que ela retorna.

```
1 def lista_de_numeros_pares(lista:list):
2     '''
3     essa função tem como objetivo percorrer a lista usando o laço for e filtrar
4     os numeros pares de uma lista qualquer usando o if
5
6     '''
7     for numero in lista:
8         if numero%2==0:
9             print(numero)
10
11
12 lista_qualquer= [1,2,3,4,5]
13
14
15
16
17 lista_de_numeros_pares(lista_qualquer)
```

# Boas Práticas

## Simplicidade

Mantenha suas funções simples e focadas em uma tarefa específica.  
Exemplo: Uma função que apenas soma dois números é simples e tem um único propósito.

## Reaproveitamento

Crie funções de forma que possam ser reutilizadas em diferentes partes do código. Exemplo: A função somar pode ser reutilizada em qualquer parte do código que necessite somar dois números.

# ATIVIDADE PRÁTICA 1

Crie uma função que receba um nome e imprima uma saudação personalizada.

# ATIVIDADE PRÁTICA 2

Crie uma função que receba um horário e imprima "Bom dia!", "Boa tarde!" ou "Boa noite!" conforme o horário.

# ATIVIDADE PRÁTICA 3

Crie uma função que receba dois números e retorne a soma deles.



# ATIVIDADE PRÁTICA 4

Crie uma função que receba dois números e retorne a subtração do primeiro pelo segundo.

# ATIVIDADE PRÁTICA 5

Crie uma função que receba dois números e retorne a multiplicação deles.

# DESAFIO PRÁTICO

## Calculadora

Crie uma calculadora com opções de soma, multiplicação, subtração, divisão e sair.

(Ela deverá funcionar infinitamente, até que o usuário decida sair da calculadora.)

Utilize funções de rotina para cada operação e funções de unidade lógica para realizar os cálculos.

### Dica:

Utilize de condicionais e Laços de Repetição para realizar esse exercício.





# SE LIGA NO CONTEÚDO DA PRÓXIMA AULA!

AULA 05 DE PYTHON.  
FUNÇÕES II.

The logo consists of the letters 'IN' in a white, bold, sans-serif font, centered within a solid red square.

**INFINITY SCHOOL**  
VISUAL ART CREATIVE CENTER



# Args (Argumentos Posicionais Arbitrários)

- A notação `*args` permite que você passe um número variável de argumentos posicionais para uma função.
- Os argumentos são coletados em uma tupla dentro da função, que pode ser acessada pelo nome `args` (ou qualquer nome de sua escolha).
- O operador `*` antes de `args` é usado para indicar que todos os argumentos posicionais a seguir devem ser coletados na tupla `args`.

# Kwargs (Argumentos de Palavra Chave)

- A notação `**kwargs` permite que você passe um número variável de argumentos de palavra-chave para uma função.
- Os argumentos de palavra-chave são coletados em um dicionário dentro da função, que pode ser acessado pelo nome `kwargs` (ou qualquer nome de sua escolha).
- O operador `**` antes de `kwargs` é usado para indicar que todos os argumentos de palavra-chave a seguir devem ser coletados no dicionário `kwargs`.



# INFINITY SCHOOL

VISUAL ART CREATIVE CENTER

AULA 04 – FUNÇÕES I