

TESTES AUTOMATIZADOS PARA WEB



PYTHON

PROF. MESTRANDO ÉRICO BORGONOVE



AGENDA

- INSTALAÇÃO UBUNTU;
- INSTALAÇÃO WINDOWS;
- AMBIENTE VISUAL STUDIO CODE0;
- CONFIGURAÇÃO DO AMBIENTE VIRTUAL;
- VARIÁVEIS
- ENTRADA E SAÍDA DE DADOS;
- OPERADORES;
- ESTRUTURAS CONDICIONAIS;
- ESTRUTURAS DE REPETIÇÃO;
- FUNÇÕES;

INSTALAÇÃO UBUNTU

Atualize os pacotes do sistema:

```
sudo apt update && sudo apt upgrade -y
```

Instale o Python 3:

```
sudo apt install python3 -y
```

Gerenciador de pacotes pip:

```
sudo apt install python3-pip -y
```

Ferramentas adicionais para
desenvolvimento:

```
sudo apt install build-essential libssl-dev libffi-dev python3-dev -y
```

Gerenciador de ambientes virtuais:

```
sudo apt install python3-venv -y
```

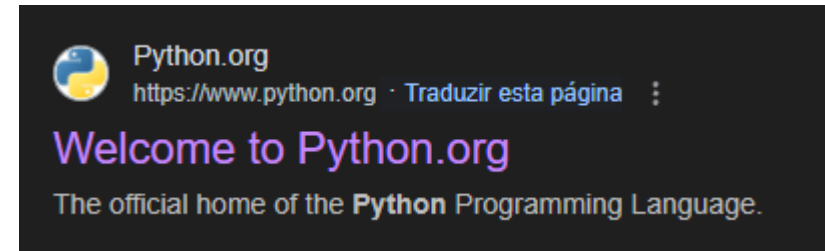
Verifique se o Python foi instalado
corretamente:

```
python3 --version  
pip3 --version
```

INSTALAÇÃO WINDOWS

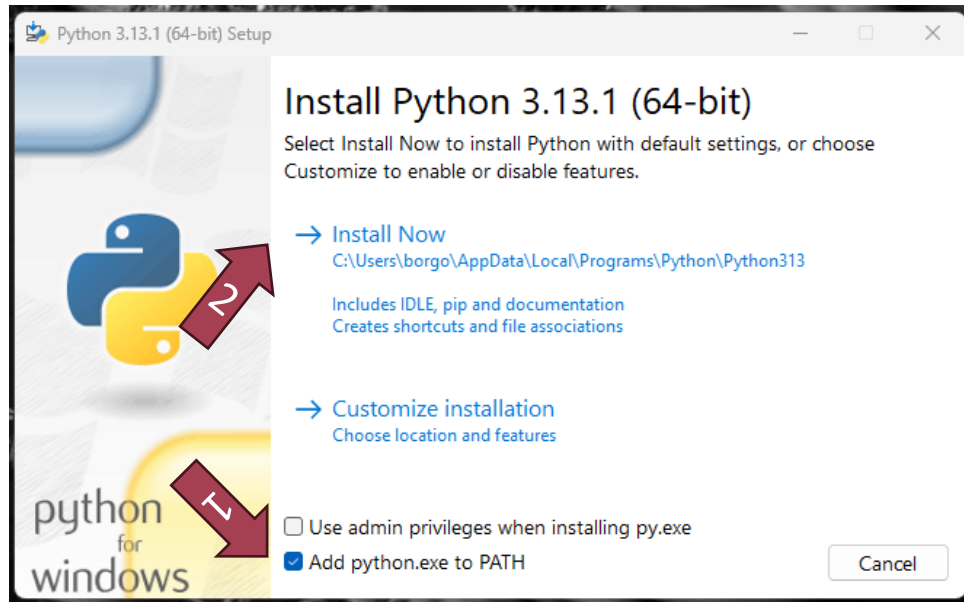
Acesse:

Faça o Download do Python 3:

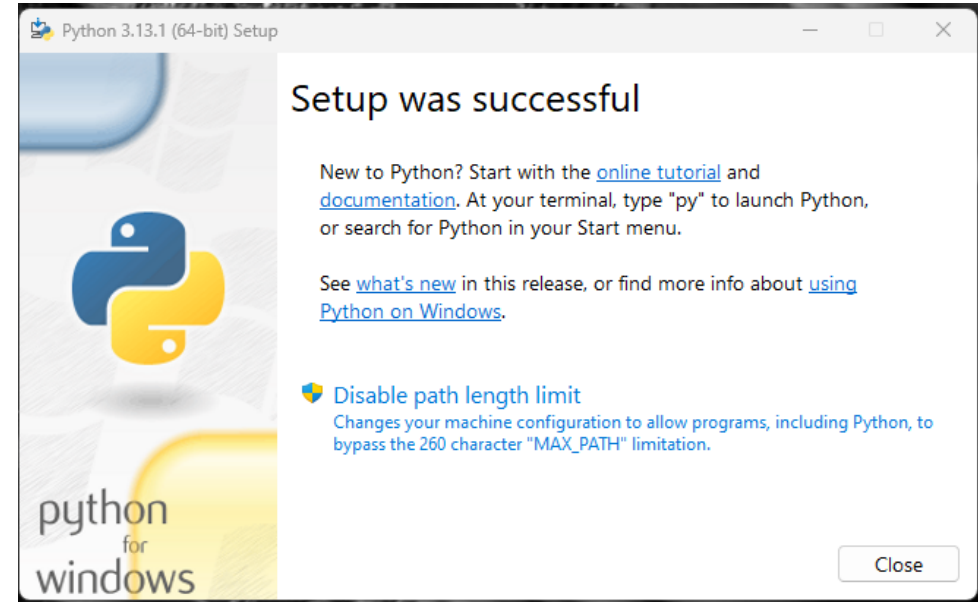


INSTALAÇÃO WINDOWS

Instale:




Finalize:




PREPARAÇÃO DO AMBIENTE NO VISUAL STUDIO CODE

Navegue até a aba de extensões e instale:




Pytest IntelliSense
Cameron Maske | 143,240 | ★★★★★ (2)
Provides autocompletion for pytest fixtures in VS Code.

[Install](#) ☒ Auto Update ⚙️




Python Extension Pack
Don Jayamanne | 10,376,321 | ★★★★★ (17)
Popular Visual Studio Code extensions for Python

[Install](#) ☒ Auto Update ⚙️



Python Snippets 3
EricSia | 254,101 | ★★★★★ (13) | ❤️ Sponsor
New auto suggestion for Python updated in 2024

[Install](#) ☒ Auto Update ⚙️



Python Extended
Taiwo Kareem | 1,665,563 | ★★★★★ (4)
Python Extended is a vscode snippet that makes it easy to write

[Install](#) ☒ Auto Update ⚙️



AMBIENTE VIRTUAL PYTHON



Escolha ou crie um diretório para o projeto:

```
mkdir meu_projeto && cd meu_projeto
```

```
mkdir meu_projeto; cd meu_projeto
```

Crie o ambiente virtual:

```
python3 -m venv venv
```

```
python -m venv venv
```

Ative o ambiente virtual:

```
source venv/bin/activate
```

```
.\venv\Scripts\Activate
```

Desative o ambiente virtual:

```
deactivate
```

VARIÁVEIS

int (inteiros): números inteiros (positivos, negativos ou zero).

```
idade = 25
```

float (ponto flutuante): números reais com casas decimais.

```
altura = 1.75
```

complex (números complexos): números na forma $a + bj$, onde a é a parte real e b é a parte imaginária.

```
numero_complexo = 3 + 4j
```

str (string): Armazena sequências de caracteres (texto). As strings podem ser delimitadas por aspas simples (') ou duplas (").

```
nome = "Python"
```

NoneType: Representa a ausência de valor ou um valor nulo.

```
resultado = None
```

As variáveis não precisam ser declaradas explicitamente com um tipo, pois Python é uma **linguagem de tipagem dinâmica**. O tipo da variável é determinado pelo valor atribuído a ela.

bool: Representa valores de verdadeiro (True) ou falso (False).

```
ativo = True
```

Para verificar o tipo de uma variável, você pode usar a função `type()`. Ela retorna o tipo da variável ou do valor atribuído a ela.

```
type(nome_da_variavel)
```

```
idade = 25
altura = 1.75
numero_complexo = 3 + 4j

print(type(idade))           # <class 'int'>
print(type(altura))          # <class 'float'>
print(type(numero_complexo)) # <class 'complex'>
```


VARIÁVEIS

list (listas): Armazena uma coleção de itens que podem ser alterados (mutáveis) e podem conter tipos diferentes.

```
frutas = ["maçã", "banana", "laranja"]
```

tuple (tuplas): Armazena uma coleção de itens, mas é imutável (não pode ser alterada).

```
coordenadas = (10, 20)
```

dict (dicionários): Armazena pares de chave e valor.

```
pessoa = {"nome": "Alice", "idade": 30}
```

set (conjuntos): Armazena uma coleção de itens únicos (sem duplicatas) em ordem não definida.

```
numeros = {1, 2, 3, 3}
```

frozenset: Igual ao set, mas é imutável.

```
numeros_congelados = frozenset([1, 2, 3])
```

```
lista = [10, 20, 30, 40]
```

```
print(lista[0]) # Primeiro elemento: 10  
print(lista[1]) # Segundo elemento: 20
```

ENTRADA E SAÍDA DE DADOS

A entrada de dados é feita usando a função `input()`, que lê dados do teclado como uma string.

- Sempre retorna uma string.
- Para outros tipos de dados (como inteiros ou floats), é necessário converter o valor lido.

```
nome = input("Digite seu nome: ") # Lê o dado como string
print(f"Olá, {nome}!")           # Saída de dados
```

```
# Para inteiros
idade = int(input("Digite sua idade: "))

# Para números com ponto flutuante
altura = float(input("Digite sua altura: "))

# Para outras conversões, como booleanos
verdade = bool(int(input("Digite 1 para True ou 0 para False: ")))
```

A saída de dados é feita principalmente com a função `print()`, que exibe informações no console.

```
print("Olá, Mundo!")
```

Você pode exibir o valor de variáveis junto com o texto:

```
nome = "Alice"
idade = 25
print(f"Meu nome é {nome} e tenho {idade} anos.") # Usando f-strings
```

Também é possível concatenar usando o operador +:

```
print("Meu nome é " + nome + " e tenho " + str(idade) + " anos.")
```

```
pi = 3.14159
print(f"O valor de pi é aproximadamente {pi:.2f}.") # Duas casas decimais
```

OPERADORES

Os operadores são símbolos ou palavras reservadas usados para realizar operações sobre valores ou variáveis. Eles podem ser divididos em diferentes categorias com base em sua funcionalidade.

OPERADORES ARITMÉTICOS

Operador	Descrição	Exemplo	Resultado
+	Adição	3+2	5
-	Subtração	3-2	1
*	Multiplicação	3*2	6
/	Divisão	3/2	1.5
//	Divisão inteira	3//2	1
%	Módulo (resto)	3%2	1
**	Exponenciação	3**2	9

OPERADORES DE COMPARAÇÃO

Operador	Descrição	Exemplo	Resultado
==	Igual a	3==3	True
!=	Diferente de	3 != 2	True
>	Maior que	3 > 2	True
<	Menor que	3 < 2	False
>=	Maior ou igual a	3 >= 2	True
<=	Menor ou igual a	3 <= 2	False

OPERADORES

OPERADORES DE ATRIBUIÇÃO

OPERADORES LÓGICOS

Operador	Descrição	Exemplo	Resultado
and	E lógico	True and False	False
or	Ou lógico	True or False	True
not	Negação lógica	not True	False

Operador	Descrição	Exemplo	Resultado
=	Atribuição simples	x = 5	--
+=	Adição e atribuição	x += 2	x = x + 2
-=	Subtração e atribuição	x -= 2	x = x - 2
*=	Multiplicação e atribuição	x *= 2	x = x * 2
/=	Divisão e atribuição	x /= 2	x = x / 2
//=	Divisão inteira e atribuição	x //= 2	x = x // 2
%=	Módulo e atribuição	x %= 2	x = x % 2
**=	Potência e atribuição	x **= 2	x = x ** 2



EXERCÍCIOS

1. Faça um programa que leia o nome de uma pessoa e mostre uma mensagem de boas vindas para ela:

Ex: Qual é o seu nome? João da Silva

Olá João da Silva, é um prazer te conhecer!

2. A locadora de carros precisa da sua ajuda para cobrar seus serviços. Escreva um programa que pergunte a quantidade de Km percorridos por um carro alugado e a quantidade de dias pelos quais ele foi alugado. Calcule o preço total a pagar, sabendo que o carro custa R\$90 por dia e R\$0,20 por Km rodado.
3. Crie um programa que leia o número de dias trabalhados em um mês e mostre o salário de um funcionário, sabendo que ele trabalha 8 horas por dia e ganha R\$ 205,52 por hora trabalhada.



CONDICIONAL

```
19. Sep 09:31 bin -> usr/bin
21. Sep 15:50 boot
19. Sep 09:32 dev
21. Sep 15:52 etc
30. Sep 2015 home
30. Sep 2015 lib -> usr/lib
30. Sep 2015 lib64 -> usr/lib
23. Jul 10:01 lost+found
1. Aug 22:45 mnt
30. Sep 2015 opt
21. Sep 15:52 private -> /home/encrypted
21. Sep 08:15 proc
12. Aug 15:37 root
21. Sep 15:50 run
30. Sep 2015 sbin -> usr/bin
30. Sep 2015 srv
21. Sep 15:51 sys
21. Sep 15:45 tmp
23. Aug 15:20
```

CONDICIONAL IF

Usado para verificar uma condição e executar um bloco de código se ela for verdadeira.

```
if condição:  
    # Código executado se a condição for verdadeira
```

```
idade = 18  
if idade >= 18:  
    print("Você é maior de idade.")
```

CONDICIONAL IF-ELSE

O else é usado para executar um bloco de código quando a condição do if for falsa.

```
if condição:  
    # Código executado se a condição for verdadeira  
else:  
    # Código executado se a condição for falsa
```

```
idade = 16  
if idade >= 18:  
    print("Você é maior de idade.")  
else:  
    print("Você é menor de idade.")
```

CONDICIONAL IF-ELIF-ELSE

O elif (abreviação de "else if") é usado para verificar múltiplas condições.

```
if condição1:
    # Código executado se a condição1 for verdadeira
elif condição2:
    # Código executado se a condição2 for verdadeira
else:
    # Código executado se nenhuma das condições anteriores for verdadeira
```

```
nota = 85

if nota >= 90:
    print("Nota A")
elif nota >= 80:
    print("Nota B")
elif nota >= 70:
    print("Nota C")
else:
    print("Nota D ou F")
```

CONDICIONAL ANINHADO

Estruturas condicionais podem ser aninhadas, ou seja, uma dentro da outra.

```
idade = 20
habilitacao = True

if idade >= 18:
    if habilitacao:
        print("Você pode dirigir.")
    else:
        print("Você precisa de uma habilitação para dirigir.")
else:
    print("Você é menor de idade e não pode dirigir.")
```


OPERADORES EM CONDICIONAIS

```
idade = 25
renda = 3000

if idade >= 18 and renda >= 2000:
    print("Você pode solicitar um cartão de crédito.")
```

```
senha = "12345"
entrada = "12345"

if entrada == senha:
    print("Acesso permitido.")
else:
    print("Acesso negado.")
```

CONDICIONAL TERNÁRIO

Python permite expressar condições simples em uma única linha, usando o operador ternário.

```
resultado = valor1 if condição else valor2
```

```
idade = 18  
status = "Maior de idade" if idade >= 18 else "Menor de idade"  
print(status)
```

VERIFICANDO VALORES COM IN E NOT IN

```
letra = 'a'

if letra in 'aeiou':
    print("É uma vogal.")
else:
    print("É uma consoante.")
```

USANDO MATCH-CASE (PYTHON 3.10+)

A partir do Python 3.10, foi introduzida a estrutura match-case, que é uma implementação nativa semelhante ao switch de outras linguagens.

```
match valor:
    case padrão1:
        # Código
    case padrão2:
        # Código
    case _:
        # Código padrão
```

```
opcao = 2

match opcao:
    case 1:
        print("Você escolheu a opção 1.")
    case 2:
        print("Você escolheu a opção 2.")
    case 3:
        print("Você escolheu a opção 3.")
    case _:
        print("Opção inválida.")
```



EXERCÍCIOS

4. Escreva um programa que pergunte a velocidade de um carro. Caso ultrapasse 80Km/h, exiba uma mensagem dizendo que o usuário foi multado. Nesse caso, exiba o valor da multa, cobrando R\$5 por cada Km acima da velocidade permitida.
5. Crie um algoritmo que leia o nome e as duas notas de um aluno, calcule a sua média e mostre na tela. No final, analise a média e mostre se o aluno teve ou não um bom aproveitamento (se ficou acima da média 7.0). E seu status (aprovado, reprovado ou recuperação).
6. Escreva um programa que leia o ano de nascimento de um rapaz e mostre a sua situação em relação ao alistamento militar.
 - Se estiver antes dos 18 anos, mostre em quantos anos faltam para o alistamento.
 - Se já tiver depois dos 18 anos, mostre quantos anos já se passaram do alistamento.





13. O Índice de Massa Corpórea (IMC) é um valor calculado baseado na altura e no peso de uma pessoa. De acordo com o valor do IMC, podemos classificar o indivíduo dentro de certas faixas.

- abaixo de 18.5: Abaixo do peso
- entre 18.5 e 25: Peso ideal
- entre 25 e 30: Sobrepeso
- entre 30 e 40: Obesidade
- acima de 40: Obseidade mórbida

Obs: O IMC é calculado pela expressão $\text{peso}/\text{altura}^2$ (peso dividido pelo quadrado da altura)

14. Um programa de vida saudável quer dar pontos atividades físicas que podem ser trocados por dinheiro. O sistema funciona assim:

- Cada hora de atividade física no mês vale pontos
- até 10h de atividade no mês: ganha 2 pontos por hora
- de 10h até 20h de atividade no mês: ganha 5 pontos por hora
- acima de 20h de atividade no mês: ganha 10 pontos por hora
- A cada ponto ganho, o cliente fatura R\$0,05 (5 centavos)

Faça um programa que leia quantas horas de atividade uma pessoa teve por mês, calcule e mostre quantos pontos ela teve e quanto dinheiro ela conseguiu ganhar.



15. Escreva um algoritmo que leia dois números inteiros e compare-os, mostrando na tela uma das mensagens abaixo:
- O primeiro valor é o maior
 - O segundo valor é o maior
 - Não existe valor maior, os dois são iguais
16. Escreva um programa para aprovar ou não o empréstimo bancário para a compra de uma casa. O programa vai perguntar o valor da casa, o salário do comprador e em quantos anos ele vai pagar. Calcule o valor da prestação mensal, sabendo que ela não pode exceder 30% do salário ou então o empréstimo será negado.
17. Uma empresa precisa reajustar o salário dos seus funcionários, dando um aumento de acordo com alguns fatores. Faça um programa que leia o salário atual, o gênero do funcionário e há quantos anos esse funcionário trabalha na empresa. No final, mostre o seu novo salário, baseado na tabela a seguir:
- | Mulheres | Homens |
|--------------------------------------|--------------------------------------|
| - menos de 15 anos de empresa: +5% | - menos de 20 anos de empresa: +3% |
| - de 15 até 20 anos de empresa: +12% | - de 20 até 30 anos de empresa: +13% |
| - mais de 20 anos de empresa: +23% | - mais de 30 anos de empresa: +25% |

REPETIÇÃO

```
19. Sep 09:31 bin -> usr/bin
21. Sep 15:50 boot
19. Sep 09:32 dev
21. Sep 15:52 etc
7 30. Sep 2015 home
7 30. Sep 2015 lib -> usr/lib
34 23. Jul 10:01 lib64 -> usr/lib
996 1. Aug 22:45 lost+found
996 30. Sep 2015 mnt
16 21. Sep 15:52 opt
0 21. Sep 08:15 private -> /home/encrypted
4096 12. Aug 15:37 proc
560 21. Sep 15:50 root
7 30. Sep 2015 run
4096 30. Sep 2015 sbin -> usr/bin
0 21. Sep 15:51 srv
300 21. Sep 15:45 sys
4096 12. Aug 15:20 tmp
```



ESTRUTURA FOR

A estrutura for é usada para iterar sobre elementos de uma sequência (listas, tuplas, strings, dicionários, ou objetos iteráveis).

```
for variavel in sequencia:  
    # Código a ser executado
```

```
frutas = ["maçã", "banana", "laranja"]  
for fruta in frutas:  
    print(fruta)
```

```
for letra in "Python":  
    print(letra)
```

```
for i in range(5): # De 0 a 4  
    print(i)  
  
for i in range(1, 6): # De 1 a 5  
    print(i)  
  
for i in range(0, 10, 2): # De 0 a 8, com passo 2  
    print(i)
```




ESTRUTURA WHILE

A estrutura while é usada para executar um bloco de código enquanto uma condição for verdadeira.

```
while condição:  
    # Código a ser executado
```

```
contador = 0  
while contador < 5:  
    print(contador)  
    contador += 1
```

```
while True:  
    print("Este é um loop infinito!")  
    break # Adicione um break para sair do loop
```


COMANDOS DE CONTROLE

Os comandos de controle são usados para alterar o fluxo natural das iterações.

Break: Usado para sair imediatamente de um laço.

```
for i in range(10):  
    if i == 5:  
        break  
    print(i)  
# Saída: 0, 1, 2, 3, 4
```

```
contador = 0  
while contador < 10:  
    if contador == 5:  
        break  
    print(contador)  
    contador += 1
```

Continue: Pula a iteração atual e passa para a próxima.

```
for i in range(5):  
    if i == 2:  
        continue  
    print(i)  
# Saída: 0, 1, 3, 4
```

```
contador = 0  
while contador < 5:  
    contador += 1  
    if contador == 3:  
        continue  
    print(contador)
```

+

○

else em Estruturas de Repetição

O bloco else pode ser usado com for ou while. Ele é executado quando o laço termina normalmente (sem break).

```
for i in range(5):  
    print(i)  
else:  
    print("Loop concluído!")
```

```
contador = 0  
while contador < 5:  
    print(contador)  
    contador += 1  
else:  
    print("Loop concluído!")
```

Quando o else não é executado

Se um break for usado para sair do laço, o bloco else não será executado.

```
for i in range(5):  
    if i == 3:  
        break  
    print(i)  
else:  
    print("Loop concluído!")  
# Saída: 0, 1, 2 (sem "Loop concluído!")
```

É possível aninhar estruturas de repetição.

```
for i in range(3):  
    for j in range(2):  
        print(f"i={i}, j={j}")
```

```
i = 0  
while i < 3:  
    j = 0  
    while j < 2:  
        print(f"i={i}, j={j}")  
        j += 1  
    i += 1
```

+

o



EXERCÍCIOS

18. Faça um algoritmo que pergunte ao usuário um número inteiro e positivo qualquer e mostre uma contagem até esse valor:

Ex: Digite um valor: 35

Contagem: 1 2 3 4 5 6 7 ... 33 34 35 Acabou!

19. Desenvolva um aplicativo que mostre na tela o resultado da expressão $500 + 450 + 400 + 350 + 300 + \dots + 50 + 0$

20. Crie um algoritmo que leia a idade de 10 pessoas, mostrando no final:

- a) Qual é a média de idade do grupo
- b) Quantas pessoas tem mais de 18 anos
- c) Quantas pessoas tem menos de 5 anos
- d) Qual foi a maior idade lida
- d) Qual foi a menor idade lida
- d) Qual foi a média das idades lida





EXERCÍCIOS

21. Desenvolva um algoritmo que leia o nome, a idade e o sexo de várias pessoas. O programa vai perguntar se o usuário quer ou não continuar. No final, mostre:
- a) O nome da pessoa mais velha
 - b) O nome da mulher mais jovem
 - c) A média de idade do grupo
 - d) Quantos homens tem mais de 30 anos
 - e) Quantas mulheres tem menos de 18 anos
22. Escreva um programa que leia um número qualquer e mostre a tabuada desse número, usando a estrutura “para”.

Ex: Digite um valor: 5

$5 \times 1 = 5$

$5 \times 2 = 10$

$5 \times 3 = 15 \dots$



FUNÇÕES

19. Sep 09:31 bin -> usr/bin
21. Sep 15:50 boot
19. Sep 09:32 dev
21. Sep 15:52 etc
30. Sep 2015 home
30. Sep 2015 lib -> usr/lib
23. Jul 10:01 lib64 -> usr/lib
1. Aug 22:45 lost+found
30. Sep 2015 mnt
21. Sep 15:52 opt
21. Sep 08:15 private -> /home/encrypted
12. Aug 15:37 proc
21. Sep 15:50 root
30. Sep 2015 run
30. Sep 2015 sbin -> usr/bin
21. Sep 15:51 srv
21. Sep 15:45 sys
22. Aug 15:20 tmp



As funções são **blocos de código reutilizáveis** que realizam uma tarefa específica. Elas permitem dividir o código em partes menores e mais organizadas, tornando o programa mais legível e reutilizável.

```
def nome_da_funcao(parametros_opcionais):  
    # Bloco de código  
    return valor_opcional
```

```
def saudacao():  
    print("Olá, mundo!")  
  
saudacao()  
# Saída: Olá, mundo!
```

Caso de teste	Cenário de Teste
Caso de teste é um conjunto de ações executadas para verificar um determinado recurso ou funcionalidade do aplicativo.	Um cenário de teste é qualquer funcionalidade do aplicativo que pode ser testada.
Caso de teste foca em como testar	O cenário de teste se concentra no que testar
O caso de teste inclui etapas de teste, dados e resultados esperados de teste	O cenário de teste inclui funcionalidades de ponta a ponta que precisam ser testadas
As equipes de controle de qualidade e desenvolvimento são responsáveis por escrever os casos de teste	Os cenários de teste são revisados por analistas de negócios ou gerentes de negócios
Isso ajuda em testes exaustivos	Isso ajuda na metodologia ágil de testar a personalidade geral
Requer mais tempo, esforço e recursos	Requer menos tempo e esforços em comparação

CASO DE TESTE X CENÁRIO DE TESTE

TIPOS DE FUNÇÕES

Funções Sem Parâmetros

Não recebem valores na chamada.

```
def exibir_mensagem():  
    print("Bem-vindo ao Python!")  
  
exibir_mensagem()
```

Funções Com Parâmetros

Recebem valores para processar.

```
def saudacao(nome):  
    print(f"Olá, {nome}!")  
  
saudacao("Alice")  
# Saída: Olá, Alice!
```

Funções Com Retorno

Usam a palavra-chave return para retornar um valor.

```
def soma(a, b):  
    return a + b  
  
resultado = soma(3, 5)  
print(resultado)  
# Saída: 8
```

FUNÇÕES ANÔNIMAS (Lambda)

Funções anônimas são criadas com a palavra-chave `lambda`. Elas são limitadas a uma única expressão.

```
lambda parametros: expressao
```

```
soma = lambda x, y: x + y  
print(soma(3, 5))  
# Saída: 8
```



EXERCÍCIOS

23. Faça um programa que possua uma função chamada `Potencia()`, que vai receber dois parâmetros numéricos (base e expoente) e vai calcular o resultado da exponenciação.

Ex: `Potencia(5,2)` vai calcular $5^2 = 25$

24. Crie uma função que receba uma string e retorne o número de vogais presentes nela.
25. Crie uma função que receba um número como argumento e retorne verdadeiro se ele for par e falso se for ímpar.
26. Crie uma função que receba um número como argumento e retorne verdadeiro se ele for positivo e falso se for negativo ou zero.
27. Crie uma função que receba dois números como argumentos e retorne o resultado da multiplicação entre eles.





REFERÊNCIAS

- Forbellone, A. L. V.; Eberspacher, H. F. Lógica de Programação – A Construção de Algoritmos e Estrutura de Dados. 2a Edição. São Paulo, Makron Books, 2000.
- Summerfield, Mark. Programação em Python 3. Uma Introdução Completa à Linguagem de Programação Python. Rio de Janeiro: Alta Books, 2012. 495 p.
- Menezes, N. N.C. Introdução à Programação com Python – 2a Ed. São Paulo, Novatec, 2014, 328 p.

