

# DESCOMPLICANDO PYTHON

**DOMINE A LINGUAGEM DE FORMA  
FÁCIL E RÁPIDA.**



**CAROLINE ALVES**



Agradeço a Deus, à  
minha família, meus  
pais, irmãos, esposo e,  
em especial aos meus  
filhos, que me dão  
forças para sempre  
buscar evoluir.

Ebook criado por Inteligência Artificial, para cumprir desafio de  
projeto em um Bootcamp realizado na plataforma DIO.

Produção:

Capa – Leonardo.ia

Conteúdo – ChatGPT

Revisão – Caroline Alves



# SUMÁRIO

01 – INTRODUÇÃO.....	05
O QUE É PYTHON.....	06
HISTÓRIA E EVOLUÇÃO.....	07
02 – FUNDAMENTOS DA PROGRAMAÇÃO EM PYTHON.....	09
INSTALAÇÃO DO PYTHON.....	10
CONFIGURAÇÃO DO AMBIENTE DE DESENV.....	13
VARIÁVEIS.....	16
TIPOS DE DADOS.....	17
OPERADORES.....	19
ESTRUTURAS DE CONDICIONAIS.....	22
LOOPS.....	23
CONTROLE DE FLUXO DENTRO DE LOOPS.....	24
FUNÇÕES.....	25
MODULARIDADE.....	28
03 – ESTRUTURA DE DADOS EM PYTHON.....	31
LISTAS.....	32
TUPLAS.....	34
CONJUNTOS.....	36
OPERAÇÕES COMUNS COM LISTAS, TUPLAS E CONJ....	38
DEFINIÇÕES E CRIAÇÃO DE STRINGS.....	40
MÉTODOS COMUNS DE STRINGS.....	42
FORMATAÇÃO DE STRINGS.....	46
04 – PROGRAMAÇÃO ORIENTADA A OBJETOS.....	48
POO – CONCEITOS BÁSICOS.....	49
MÉTODOS.....	49
05 – TRATAMENTO DE EXCEÇÕES.....	55
EXCEÇÕES EM PYTHON.....	56
LEVANTAMENTO DE EXCEÇÕES.....	58



ERROS COMUNS EM PYTHON.....	59
TRATAMENTO DE EXCEÇÕES GENÉRICAS.....	60
06 – MÓDULOS E PACOTES.....	61
IMPORTANDO MÓDULOS.....	62
IMPORTAÇÃO ESPECÍFICA DE FUNÇÕES OU CLASSES...	63
RENAMEANDO MÓDULOS E FUNÇÕES.....	64
IMPORTANDO TODOS OS ITENS DE UM MÓDULO.....	65
CRIANDO E IMPORTANDO SEUS PRÓPRIOS MÓDULOS.	66
IMPORTANDO PACOTES.....	66
BIBLIOTECAS PADRÃO E EXTERNAS.....	67
BIBLIOTECAS EXTERNAS.....	68
REFERÊNCIAS.....	70



# 01

## INTRODUÇÃO

## - O que é Python?

É uma linguagem de programação de alto nível (assim chamada por ser mais próxima da linguagem humana), conhecida por sua simplicidade e facilidade de uso.

É uma linguagem interpretada, o que significa que seu código é executado linha por linha pelo interpretador. Python é o que chamamos de linguagem multiparadigma, podendo ser usada para desenvolver uma ampla variedade de aplicações, desde scripts simples até grandes sistemas complexos.

Em comparação com outras linguagens, como Java, ou C++, o Python requer menos linhas de código para realizar a mesma tarefa, o que reduz a chance de erros e torna o processo de desenvolvimento mais rápido.

Python possui bibliotecas que podem ser utilizadas desde o desenvolvimento web, até o aprendizado de máquina e inteligência artificial.

## - História e evolução:

Sua história se iniciou nos anos de 1980, quando foi criado por Guido van Rossum. Ele trabalhava no Instituto de Pesquisa Nacional para Matemática e Ciência da Computação na Holanda (CWI). A ideia inicial era criar uma linguagem que pudesse preencher as lacunas deixadas pela linguagem ABC, a qual ele havia ajudado a desenvolver.

Em 1991 foi lançada a primeira versão do Python (0.9), e em 1994, introduzido novos recursos importantes (1.0), como a modulação do sistema de importação, o suporte para programação funcional e a estrutura de dados lambda. Ficou popular em 1995, quando mais e mais desenvolvedores começaram a adotar a linguagem para projetos web.

Em 2000, foi lançada a versão 2.0, com diversas melhorias e suporte para Unicode.

Em 2008, a versão 3.0 de Python foi projetada para corrigir falhas fundamentais e melhorar a consistência da linguagem.

---

Em 2010, veio o Python 3.1, com melhorias da linguagem de programação e novos recursos.

Em 2015, a versão 3.5 introduziu o operador de matrizes '@' e as *async/await*, que facilitam a programação assíncrona.

A versão 3.7 chegou em 2018, trazendo novos recursos como *dataclasses*, que simplificam a criação de classes para armazenar dados, e melhorias significativas no desempenho.

A evolução continua, e em 2021, o lançamento da versão 3.9, introduz melhorias na sintaxe e desempenho, como a nova sintaxe para união de tipos (Union) e o suporte melhorado para programação paralela.

A versão mais atual (3.11), foi lançada em 2023, adicionando novos recursos e melhorando a eficiência e a usabilidade da linguagem.





# 02

## FUNDAMENTOS DA PROGRAMAÇÃO COM PYTHON

---

# Instalação do Python

## Passo 1: Baixar Python

Acesse o site oficial do Python: Vá para [python.org](https://python.org).

Baixe a versão mais recente: Na página principal, clique em "Downloads" e selecione a versão adequada para o seu sistema operacional (Windows, macOS ou Linux). Para a maioria dos usuários, a versão recomendada será a última versão estável.

## Passo 2: Instalar Python

### Windows:

Execute o instalador que você baixou.

Certifique-se de marcar a opção *"Add Python to PATH"* para que o Python seja acessível a partir da linha de comando.

Clique em *"Install Now"* e siga as instruções na tela.



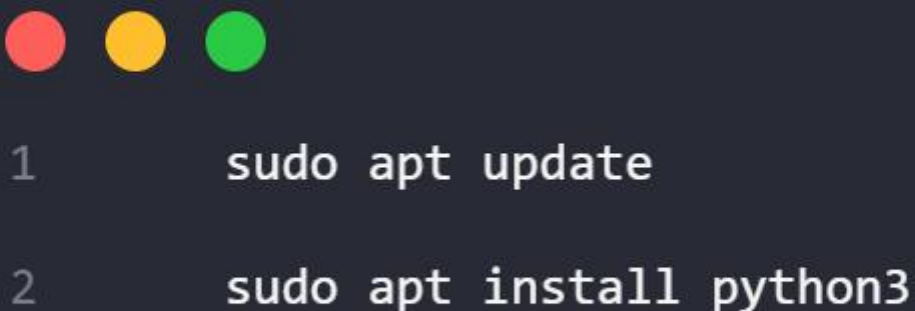
## macOS:

Execute o instalador *.pkg* que você baixou.

Siga as instruções na tela para concluir a instalação.

## Linux:

Em distribuições baseadas em Debian (como Ubuntu), você pode instalar o Python usando o comando:



```
1      sudo apt update
2      sudo apt install python3
```

Para outras distribuições, consulte a documentação específica.

## Verificar a Instalação

Depois de instalar, você pode verificar se o Python está corretamente instalado abrindo o terminal (ou prompt de comando no Windows) e digitando:



Isso deve retornar a versão do Python instalada.



# Configuração do Ambiente de Desenvolvimento

Um ambiente de desenvolvimento integrado (IDE) facilita a escrita, execução e depuração do código. Aqui estão algumas opções populares:

## Visual Studio Code (VS Code):

Download e Instalação: Acesse [code.visualstudio.com](https://code.visualstudio.com) e baixe o instalador para o seu sistema operacional.

Configuração: Após instalar o VS Code, instale a extensão "Python" da Microsoft. Isso adicionará suporte para Python, incluindo realce de sintaxe, autocompletar, linting, depuração, entre outros.

Extensões: Além da extensão Python, você pode instalar outras extensões úteis como Pylance (para inteligência artificial e sugestões de código), Jupyter (para notebooks interativos), e GitLens (para controle de versão).

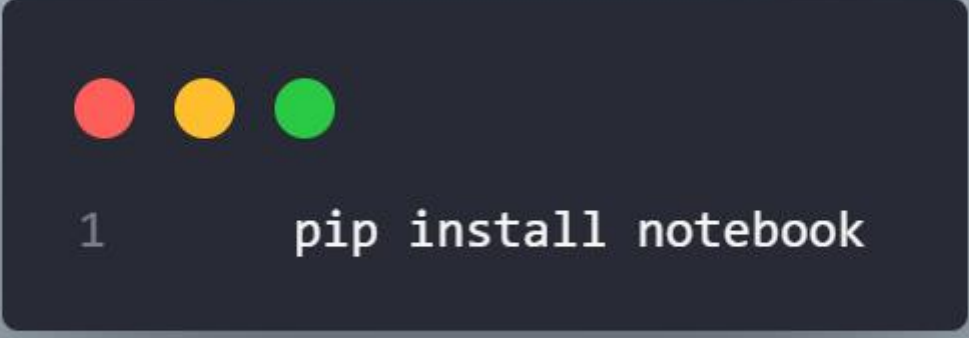
## PyCharm:

Download e Instalação: Acesse [jetbrains.com/pycharm](https://jetbrains.com/pycharm) e baixe a versão Community (gratuita) ou Professional (paga).

Configuração: Siga as instruções do instalador. PyCharm é um IDE especializado para Python, com recursos avançados para desenvolvimento, depuração e teste de código.

## Jupyter Notebook:

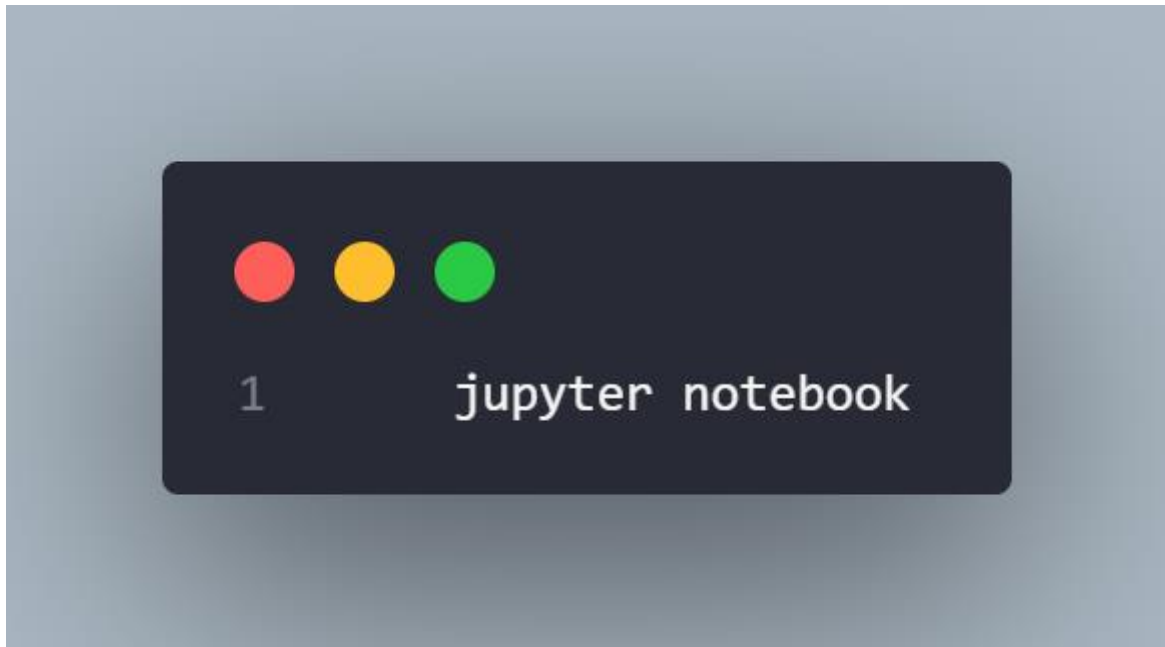
Instalação: Jupyter Notebooks são populares para ciência de dados e aprendizado de máquina. Você pode instalá-los usando o comando:

A terminal window with a dark background and three colored window control buttons (red, yellow, green) at the top left. The text '1 pip install notebook' is displayed in a light gray font.

```
1 pip install notebook
```



Uso: Após a instalação, você pode iniciar um notebook com o comando:

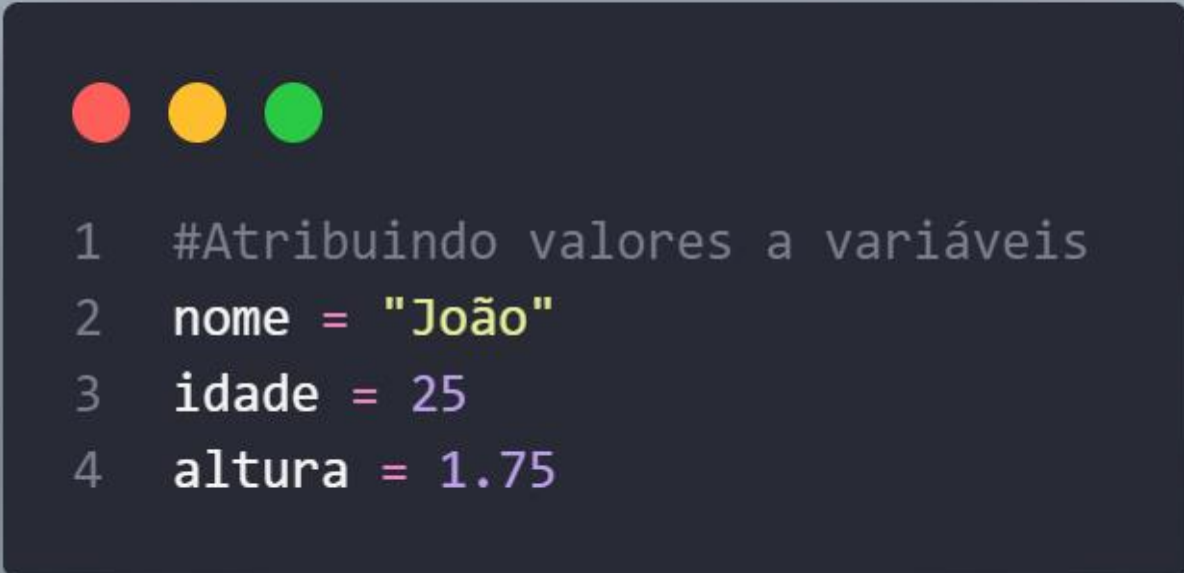


Isso abrirá uma interface web onde você pode criar e executar notebooks interativos.

## Variáveis:

Variáveis são usadas para armazenar valores que podem ser usados e manipulados no seu programa. Pense em variáveis como caixas que guardam informações. Em Python, você não precisa declarar o tipo da variável antes de usá-la; você simplesmente atribui um valor a ela.

### Exemplo:


A screenshot of a code editor window with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. The code is written in Python and shows four lines of variable assignment. The first line is a comment. The second line assigns the string "João" to the variable 'nome'. The third line assigns the integer 25 to the variable 'idade'. The fourth line assigns the float 1.75 to the variable 'altura'.

```
1  #Atribuindo valores a variáveis
2  nome = "João"
3  idade = 25
4  altura = 1.75
```



## Tipos de Dados:

Python possui vários tipos de dados que você pode usar para armazenar diferentes tipos de informações. Aqui estão alguns dos tipos de dados mais comuns:



```
1  # Inteiros (int): Números inteiros,  
2  # positivos ou negativos, sem ponto decimal.  
3  idade = 25  
4  
5  # Ponto Flutuante (float):  
6  # Números reais, com ponto decimal.  
7  altura = 1.75  
8  
9  # String (str):  
10 # Sequências de caracteres,  
11 # usadas para representar texto.  
12 nome = "João"
```



```
1  # Booleano (bool):
2  # Representa valores lógicos,
3  # True ou False.
4  estudante = True
5
6  # Listas (list):
7  # Coleções ordenadas de itens,
8  # que podem ser
9  # de diferentes tipos.
10 frutas = ["maçã", "banana", "laranja"]
11
12 # Tuplas (tuple): Semelhantes às listas,
13 # mas imutáveis (não podem ser
14 # alteradas após a criação).
15 coordenadas = (10, 20)
16
17 # Dicionários (dict):
18 # Coleções de pares
19 # chave-valor, usadas para
20 # armazenar dados relacionados.
21 aluno = {"nome": "João", "idade": 25, "altura": 1.75}
```

# Operadores:

Operadores são símbolos que realizam operações em valores e variáveis. Aqui estão alguns dos operadores mais comuns em Python:

## Operadores aritméticos:

```
1  # Adição (+): Soma valores.
2  resultado = 10 + 5  # 15
3
4  # Subtração (-): Subtrai um valor de outro.
5  resultado = 10 - 5  # 5
6
7  # Multiplicação (*): Multiplica valores.
8  resultado = 10 * 5  # 50
9
10 # Divisão (/): Divide um valor por outro,
11 # retornando um float.
12 resultado = 10 / 5  # 2.0
13
14 # Divisão Inteira (//): Divide um valor por
15 # outro, retornando um inteiro.
16 resultado = 10 // 3  # 3
17
18 # Módulo (%): Retorna o resto da divisão.
19 resultado = 10 % 3  # 1
20
21 # Exponenciação (**): Eleva um valor à
22 # potência de outro.
23 resultado = 2 ** 3  # 8
```



## Operadores de comparação:



```
1  # Igual (==): Verifica se
2  # dois valores são iguais.
3  resultado = 10 == 10  # True
4
5  # Diferente (!=): Verifica
6  # se dois valores são diferentes.
7  resultado = 10 != 5   # True
8
9  # Maior que (>): Verifica se
10 # um valor é maior que outro.
11 resultado = 10 > 5    # True
12
13 # Menor que (<): Verifica se
14 # um valor é menor que outro.
15 resultado = 10 < 5    # False
16
17 # Maior ou igual (>=): Verifica
18 # se um valor é maior ou igual a outro.
19 resultado = 10 >= 10  # True
20
21 # Menor ou igual (<=): Verifica
22 # se um valor é menor ou igual a outro.
23 resultado = 10 <= 5   # False
```

## Operadores lógicos:



```
1  # E (and): Retorna True se ambas as
2  # expressões forem verdadeiras.
3  resultado = (10 > 5) and (10 < 20) # True
4
5  # Ou (or): Retorna True se pelo
6  # menos uma das expressões for verdadeira.
7  resultado = (10 > 5) or (10 > 20) # True
8
9  # Não (not): Inverte o valor
10 # lógico da expressão.
11 resultado = not (10 > 5) # False
```

## Estruturas Condicionais


As estruturas condicionais permitem que você execute diferentes blocos de código com base em certas condições.

*if, elif, else*

'if': Executa um bloco de código se uma condição for verdadeira.

'elif': (abreviação de "else if") Executa um bloco de código se a condição anterior for falsa e a nova condição for verdadeira.

'else': Executa um bloco de código se todas as condições anteriores forem falsas.



```
1  idade = 20
2
3  if idade < 18:
4      print("Menor de idade")
5  elif idade >= 18 and idade < 60:
6      print("Adulto")
7  else:
8      print("Idoso")
```



## Loops

Os loops permitem que você repita a execução de um bloco de código várias vezes.

*'for'*

O loop for é usado para iterar sobre uma sequência (como uma lista, tupla ou string) ou qualquer outro objeto iterável.

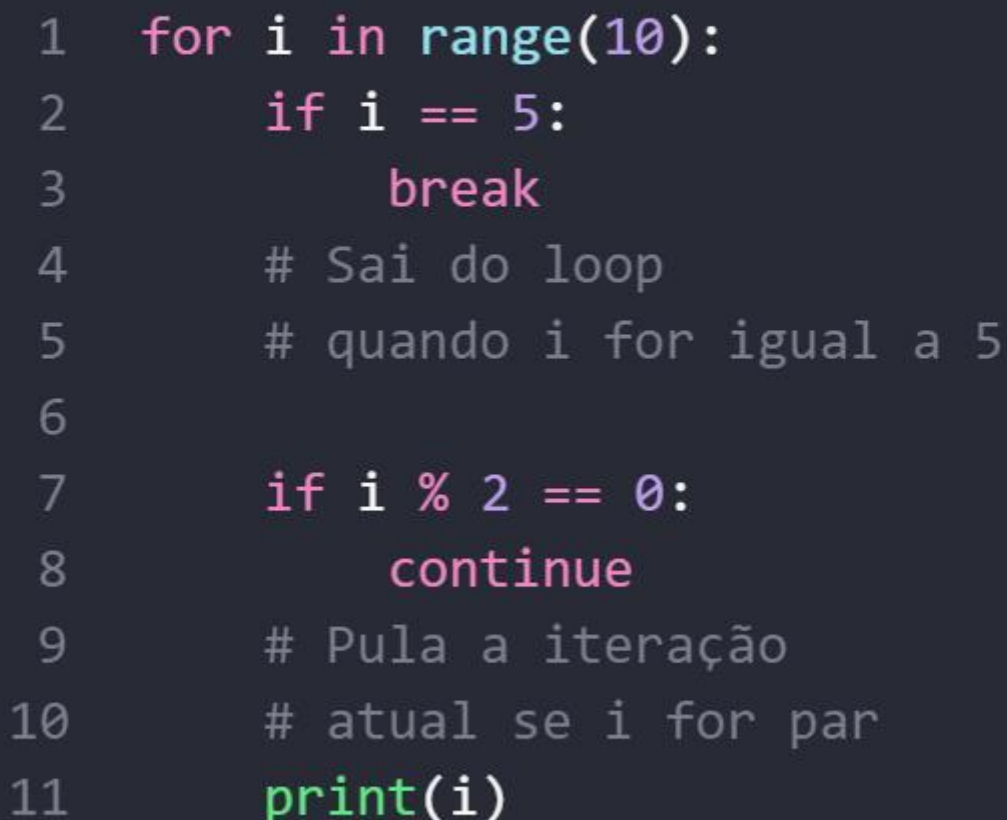


```
1  # Exemplo com for:
2  frutas = ["maçã", "banana", "laranja"]
3
4  for fruta in frutas:
5      print(fruta)
6
7  # Exemplo com range:
8  for i in range(5):
9      print(i)
10
11 # Exemplo com while:
12 contador = 0
13
14 while contador < 5:
15     print(contador)
16     contador += 1
```

## Controle de Fluxo Dentro de Loops

'*break*': Sai imediatamente do loop.

'*continue*': Pula a iteração atual e vai para a próxima iteração do loop.



```
1  for i in range(10):
2      if i == 5:
3          break
4      # Sai do loop
5      # quando i for igual a 5
6
7      if i % 2 == 0:
8          continue
9      # Pula a iteração
10     # atual se i for par
11     print(i)
```


Agora vamos ver conceitos fundamentais que ajudam a organizar e reutilizar o código de maneira eficiente e clara.

## Funções:

Funções são blocos de código que realizam uma tarefa específica e podem ser reutilizadas em diferentes partes do programa. Elas permitem dividir um programa grande em pedaços menores e mais gerenciáveis.

Definindo Funções: Para definir uma função em Python, você usa a palavra-chave `'def'`, seguida pelo nome da função, parênteses e dois pontos. O código da função é escrito em um bloco indentado abaixo da definição.

Exemplo:



```
1  def saudacao():
2      print("Olá, Mundo!")
3
4  # Chamando a função
5  saudacao()
```



Funções com Parâmetros: Funções podem aceitar parâmetros, que são valores fornecidos à função quando ela é chamada. Esses parâmetros permitem que a função seja mais flexível e útil.

Exemplo:

A screenshot of a code editor with a dark background and three colored window control buttons (red, yellow, green) at the top left. The code is written in Python and consists of four lines, each preceded by a line number (1, 2, 3, 4). The code defines a function named 'saudacao\_com\_nome' that takes a parameter 'nome' and prints a greeting. It then calls this function with the argument 'João'.

```
1 def saudacao_com_nome(nome):  
2     print(f"Olá, {nome}!")  
3  
4 saudacao_com_nome("João")
```

Funções com Retorno: Funções podem retornar valores usando a palavra-chave '*return*'. Isso permite que a função produza um resultado que pode ser usado em outras partes do programa.

Exemplo:



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

Parâmetros Padrão: Você pode definir valores padrão para os parâmetros de uma função. Se o parâmetro não for fornecido quando a função é chamada, o valor padrão será usado.

Exemplo:



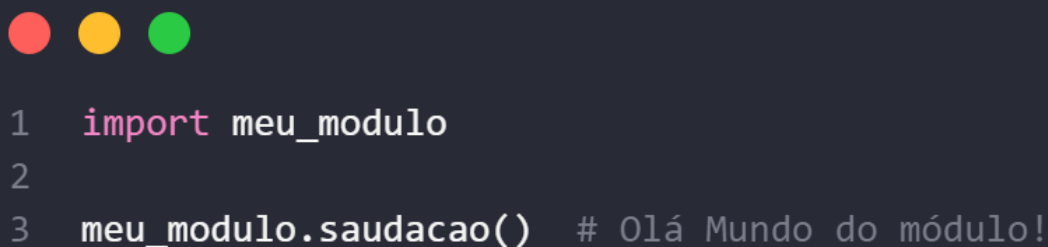
```
1 def saudacao_com_nome(nome="Mundo"):
2     print(f"Olá, {nome}!")
3
4 saudacao_com_nome() # Olá, Mundo!
5 saudacao_com_nome("João") # Olá, João!
```

## Modularidade

A modularidade refere-se à prática de dividir o código em módulos menores e independentes, que podem ser combinados para criar um sistema maior. Em Python, você pode criar módulos e pacotes para organizar seu código.

**Módulos:** Um módulo é um arquivo Python que contém funções, variáveis e classes que podem ser reutilizados em outros arquivos Python. Para usar um módulo, você pode importá-lo usando a palavra-chave *'import'*.

**Exemplo:** Suponha que você tenha um arquivo chamado *'meu\_modulo.py'* com o seguinte conteúdo. Você pode importar este módulo em outro arquivo Python:

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. It displays three lines of Python code with line numbers 1, 2, and 3 on the left. The code imports a module and calls a function.

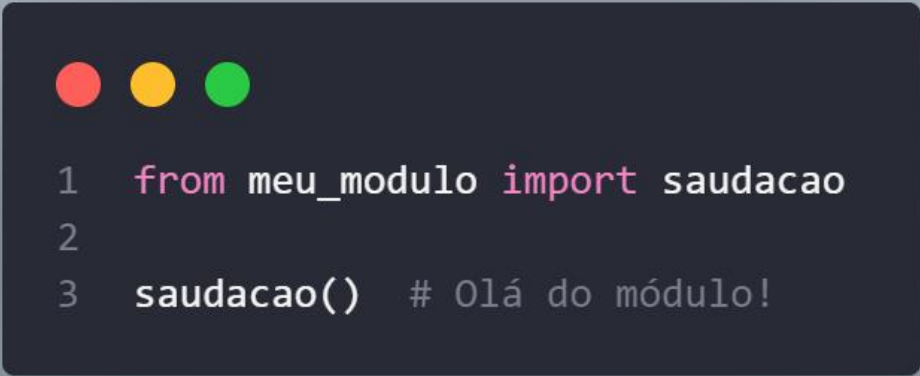
```
1  import meu_modulo
2
3  meu_modulo.saudacao()  # Olá Mundo do módulo!
```



## Importações Específicas:

Você pode importar apenas partes específicas de um módulo usando a palavra-chave *'from'*.


Exemplo:



```
1 from meu_modulo import saudacao
2
3 saudacao() # Olá do módulo!
```

Pacotes: Um pacote é uma coleção de módulos organizados em diretórios que contêm um arquivo especial chamado *'\_\_init\_\_.py'*. Este arquivo pode estar vazio, mas sua presença indica que o diretório deve ser tratado como um pacote Python.

## Estrutura de Pacotes:



```
1 meu_pacote/
2     __init__.py
3     modulo1.py
4     modulo2.py
```

---

## Vantagens da Modularidade:

Reutilização de Código: Módulos e funções podem ser reutilizados em diferentes partes do programa ou em outros projetos.

Manutenção Facilitada: Código modular é mais fácil de manter e atualizar, pois mudanças em um módulo específico não afetam o restante do programa.

Legibilidade: Dividir o código em funções e módulos torna o programa mais fácil de entender e seguir.

Organização: Pacotes ajudam a organizar o código em uma estrutura lógica, facilitando a navegação e localização de funcionalidades específicas.

Funções e modularidade são técnicas poderosas que ajudam a escrever código mais organizado, reutilizável e fácil de manter. Ao usar funções para dividir tarefas específicas e módulos para organizar o código, você pode criar programas mais eficientes e robustos.

# 03

## ESTRUTURAS DE DADOS EM PYTHON

---

Para iniciar esse módulo, vamos falar sobre como armazenar coleções de elementos.

## Listas

As listas são coleções ordenadas de elementos que são mutáveis, o que significa que você pode alterar, adicionar e remover itens depois que a lista for criada. Listas são definidas usando colchetes '[' ]'.

### Características das Listas

Ordenadas: A ordem dos elementos é mantida.

Mutáveis: Elementos podem ser alterados, adicionados ou removidos.

Permitem duplicatas: Podem conter elementos repetidos.

A seguir veremos exemplos de listas e como elas são criadas em Python:





```
1  # Criando uma lista
2  frutas = ["maçã", "banana", "laranja"]
3
4  # Acessando elementos
5  print(frutas[0])
6  # maçã
7
8  # Alterando elementos
9  frutas[1] = "uva"
10 print(frutas)
11 # ["maçã", "uva", "laranja"]
12
13 # Adicionando elementos
14 frutas.append("manga")
15 print(frutas)
16 # ["maçã", "uva", "laranja", "manga"]
17
18 # Removendo elementos
19 frutas.remove("uva")
20 print(frutas)
21 # ["maçã", "laranja", "manga"]
```

# Tuplas

As tuplas são coleções ordenadas de elementos que são imutáveis, ou seja, uma vez criada, a tupla não pode ser alterada.

Tuplas são definidas usando parênteses '()'.

## Características das Tuplas

Ordenadas: A ordem dos elementos é mantida.

Imutáveis: Não podem ser alteradas após a criação.

Permitem duplicatas: Podem conter elementos repetidos.

A seguir veja exemplos de tuplas e como elas são usadas em Python:



```
1  # Criando uma tupla
2  coordenadas = (10, 20)
3
4  # Acessando elementos
5  print(coordenadas[0])
6  # 10
7
8  # 1 - Tentando alterar
9  #     um elemento
10 #     (resultará em erro)
11 # 2 - coordenadas[0] = 15
12 # 3 - TypeError: 'tuple'
13 #     object does not
14 #     support item assignment
15
16 # Tuplas podem ser
17 # usadas como chaves em
18 # dicionários, enquanto
19 # listas não podem.
20 ponto = {coordenadas: "Ponto A"}
21 print(ponto)
22 # {(10, 20): "Ponto A"}
```

# Conjuntos

Os conjuntos são coleções não ordenadas de elementos únicos. Conjuntos são mutáveis, mas não permitem elementos duplicados.

Conjuntos são definidos usando chaves '{ }' ou a função 'set()'.

## Características dos Conjuntos:

Não ordenados: A ordem dos elementos não é garantida.

Mutáveis: Elementos podem ser adicionados ou removidos.

Não permitem duplicatas: Cada elemento é único.

A seguir veja exemplos de Conjuntos e como são utilizados em Python:





```
1  # Criando um conjunto
2  frutas = {"maçã", "banana", "laranja"}
3
4  # Adicionando elementos
5  frutas.add("uva")
6  print(frutas)
7  # A ordem dos elementos pode variar
8
9  # Removendo elementos
10 frutas.remove("banana")
11 print(frutas)
12 # A ordem dos elementos pode variar
13
14 # Tentando adicionar duplicatas (não terá efeito)
15 frutas.add("maçã")
16 print(frutas)
17 # {"maçã", "laranja", "uva"}
18
```

# Operações Comuns com Listas, Tuplas e Conjuntos

## Listas:


- Adicionar elementos:
  - `'append( )'`, `'extend( )'`, `'insert( )'`
- Remover elementos:
  - `'remove( )'`, `'pop( )'`, `'clear( )'`
- Ordenar:
  - `'sort( )'`, `'reverse( )'`

## Tuplas:

- Indexação e fatiamento:
  - Igual às listas
- Operações com tuplas:
  - Concatenar `'+'`, repetir `'*'`

## Conjuntos:

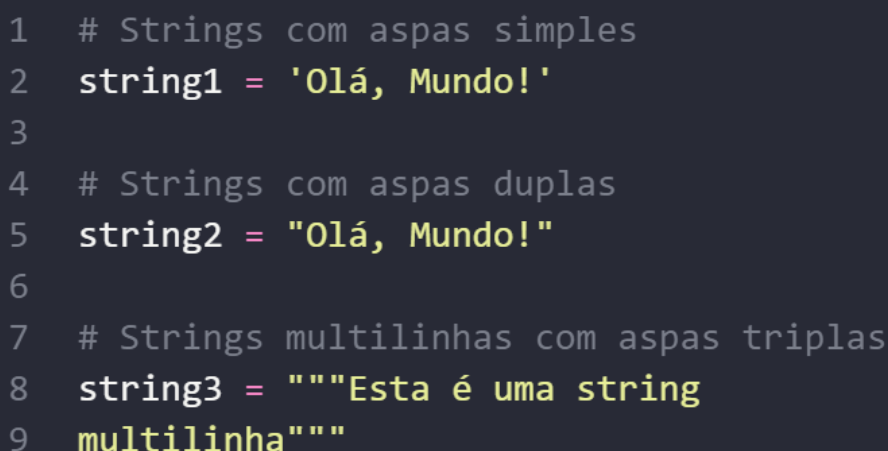
- Operações de conjunto:
  - `'union( )'`, `'intersection( )'`, `'difference( )'`, `'symmetric_difference( )'`
- Adicionar elementos:
  - `'add( )'`, `'update( )'`
- Remover elementos:
  - `'remove( )'`, `'discard( )'`, `'clear( )'`



```
1  # Listas
2  numeros = [1, 2, 3]
3  numeros.append(4)
4  print(numeros)
5  # [1, 2, 3, 4]
6
7  # Tuplas
8  tupla1 = (1, 2, 3)
9  tupla2 = (4, 5)
10 tupla3 = tupla1 + tupla2
11 print(tupla3)
12 # (1, 2, 3, 4, 5)
13
14 # Conjuntos
15 conjunto1 = {1, 2, 3}
16 conjunto2 = {3, 4, 5}
17 uniao = conjunto1.union(conjunto2)
18 print(uniao)
19 # {1, 2, 3, 4, 5}
```

## Definição e Criação de Strings

Strings em Python são definidas usando aspas simples (') ou duplas (").



```
1 # Strings com aspas simples
2 string1 = 'Olá, Mundo!'
3
4 # Strings com aspas duplas
5 string2 = "Olá, Mundo!"
6
7 # Strings multilinhas com aspas triplas
8 string3 = """Esta é uma string
9 multilinha"""
```

## Operações Básicas com Strings

### Concatenação:

A concatenação é a operação de juntar duas ou mais strings usando o operador '+'.

### Repetição:

Você pode repetir uma string usando o operador '\*'.



## Indexação e Fatiamento:

Você pode acessar caracteres individuais de uma string usando a indexação ('[ ]') e obter substrings usando o fatiamento ('[inicio:fim:passo]').

```
1  # Exemplo Concatenação:
2  saudacao = "Olá"
3  nome = "João"
4  mensagem = saudacao + ", " + nome + "!"
5  print(mensagem)  # Olá, João!
6
7  # Exemplo Repetição:
8  repetir = "Olá! " * 3
9  print(repetir)  # Olá! Olá! Olá!
10
11 texto = "Python"
12 # Indexação
13 print(texto[0])  # P
14 print(texto[-1])  # n
15
16 # Fatiamento
17 print(texto[0:2])  # Py
18 print(texto[2:])  # thon
19 print(texto[:2])  # Py
20 print(texto[::2])  # Pto (cada 2 caracteres)
```

# Métodos Comuns de Strings

Python fornece muitos métodos embutidos para manipular strings.


## Mudança de Caso

'*upper()*': Converte a string para maiúsculas.

'*lower()*': Converte a string para minúsculas.

'*title()*': Converte a primeira letra de cada palavra para maiúscula.

'*capitalize()*': Converte a primeira letra da string para maiúscula.



```
1 texto = "python é incrível"
2 print(texto.upper()) # PYTHON É INCRÍVEL
3 print(texto.lower()) # python é incrível
4 print(texto.title()) # Python É Incrível
5 print(texto.capitalize()) # Python é incrível
```

## Remoção de Espaços

`'strip( )'`: Remove espaços em branco do início e do fim da string.

`'lstrip( )'`: Remove espaços em branco do início da string.


`'rstrip( )'`: Remove espaços em branco do fim da string.

## Substituição e Divisão

`'replace(antigo, novo)'`: Substitui todas as ocorrências de uma substring por outra.

`'split(separador)'`: Divide a string em uma lista de substrings com base em um separador.

`'join(iterável)'`: Junta uma lista de strings em uma única string, usando um separador especificado.



```
1  # Remoção de espaços
2  texto = "    Olá, Mundo!    "
3  print(texto.strip())
4  # "Olá, Mundo!"
5  print(texto.lstrip())
6  # "Olá, Mundo!    "
7  print(texto.rstrip())
8  # "    Olá, Mundo!"
9
10 texto = "Olá, Mundo! Mundo é incrível"
11 # Substituição
12 novo_texto = texto.replace("Mundo", "Python")
13 print(novo_texto)
14 # Olá, Python! Python é incrível
15
16 # Divisão
17 palavras = texto.split(" ")
18 print(palavras)
19 # ['Olá,', 'Mundo!', 'Mundo', 'é', 'incrível']
20
21 # Junção
22 frase = " ".join(palavras)
23 print(frase)
24 # Olá, Mundo! Mundo é incrível
```



## Verificação de Conteúdo

'*startswith(prefixo)*': Verifica se a string começa com o prefixo especificado.

'*endswith(sufixo)*': Verifica se a string termina com o sufixo especificado.

'*in*': Verifica se uma substring está presente na string.

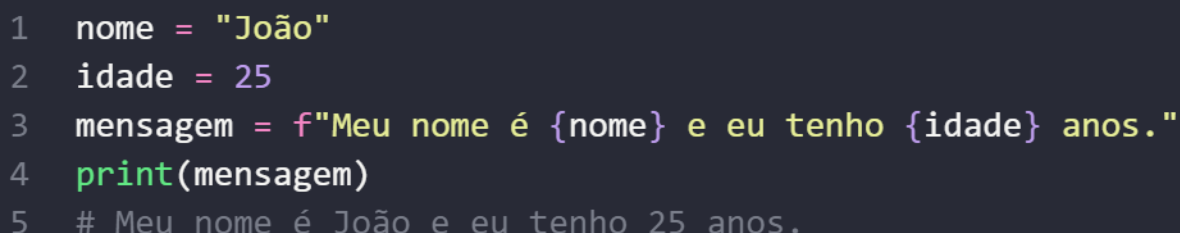
A screenshot of a code editor window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. The code is written in Python and demonstrates string verification methods. It includes line numbers from 1 to 13 on the left side of the code block.

```
1 texto = "Python é divertido"
2
3 # Verificação de início e fim
4 print(texto.startswith("Python"))
5 # True
6 print(texto.endswith("divertido"))
7 # True
8
9 # Verificação de conteúdo
10 print("divertido" in texto)
11 # True
12 print("chato" in texto)
13 # False
```

## Formatação de Strings

A formatação de strings permite incluir variáveis e expressões dentro de uma string de maneira elegante e legível. Existem várias maneiras de fazer isso em Python.

*'f-strings'*(Format Strings): Introduzidas no Python 3.6, as *'f-strings'* são uma maneira concisa e eficiente de formatar strings.

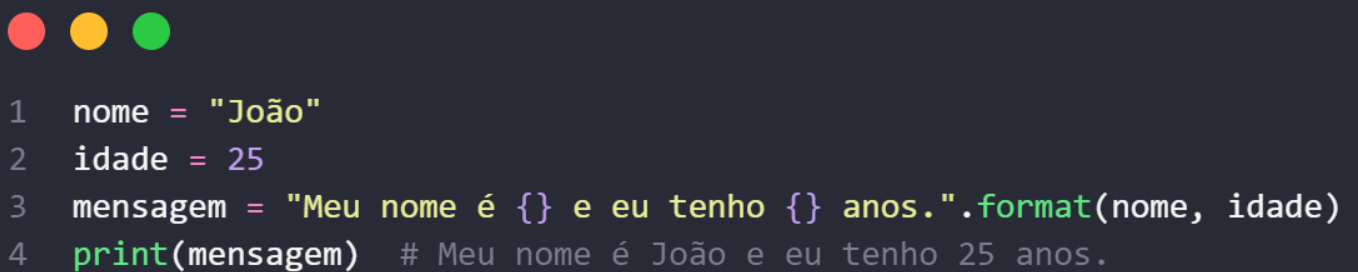
A code editor window with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. It contains five lines of Python code. The first line assigns 'João' to the variable 'nome'. The second line assigns 25 to the variable 'idade'. The third line uses an f-string to create a message: 'Meu nome é {nome} e eu tenho {idade} anos.'. The fourth line prints the message. The fifth line is a comment showing the resulting string.

```
1 nome = "João"
2 idade = 25
3 mensagem = f"Meu nome é {nome} e eu tenho {idade} anos."
4 print(mensagem)
5 # Meu nome é João e eu tenho 25 anos.
```

Método *'format( )'*

Outra maneira de formatar strings é usando o método *'format( )'*.

Exemplo:



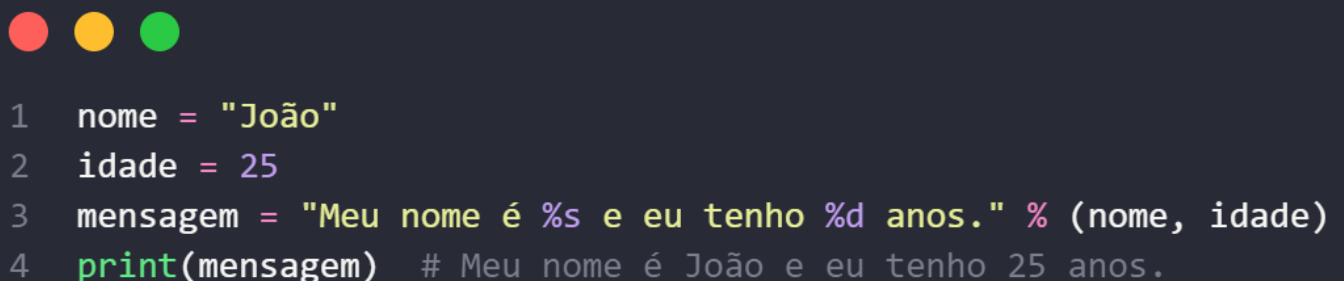
```

1  nome = "João"
2  idade = 25
3  mensagem = "Meu nome é {} e eu tenho {} anos.".format(nome, idade)
4  print(mensagem)  # Meu nome é João e eu tenho 25 anos.
    
```

## Operador ‘%’

Uma maneira mais antiga de formatar strings é usando o operador ‘%’.

Exemplo:



```

1  nome = "João"
2  idade = 25
3  mensagem = "Meu nome é %s e eu tenho %d anos." % (nome, idade)
4  print(mensagem)  # Meu nome é João e eu tenho 25 anos.
    
```

Desde operações básicas como concatenação e fatiamento até métodos mais avançados de formatação e substituição, a manipulação de strings é uma parte fundamental do desenvolvimento de software em Python.

# 04

## PROGRAMAÇÃO ORIENTADA A OBJETOS



# Programação Orientada a Objetos (POO):

## Conceitos Básicos:

### 1 – Classe:

- Uma classe é uma definição ou um molde que descreve o comportamento e os atributos que os objetos dessa classe terão.
- Em Python, uma classe é definida usando a palavra-chave `'class'`.

### 2 – Objeto:

- Um objeto é uma instância de uma classe. Ele representa uma entidade concreta que possui estado e comportamento definidos pela classe.

### 3 – Atributo:

- Atributos são variáveis que pertencem a uma classe ou a uma instância de classe. Eles representam o estado ou as propriedades de um objeto.

## Método:

- Métodos são funções definidas dentro de uma classe que descrevem os

---

comportamentos que um objeto pode realizar. Eles podem manipular os atributos do objeto ou executar outras ações.

### 1 – Encapsulamento:

- O encapsulamento é o conceito de esconder os detalhes internos de um objeto e fornecer uma interface pública controlada para interagir com ele. Isso ajuda a proteger os dados e a manter a integridade do objeto.

### 2 – Herança:

- A herança é um mecanismo pelo qual uma classe pode herdar atributos e métodos de outra classe. A classe que herda é chamada de subclasse, e a classe da qual ela herda é chamada de superclasse.

### 3 – Polimorfismo:

- O polimorfismo permite que objetos de diferentes classes sejam tratados como objetos de uma classe comum. Isso é geralmente alcançado através da herança e da sobrecarga de métodos.



```
1  # Definindo uma Classe
2  class Animal:
3      def __init__(self, nome, idade):
4          self.nome = nome
5          # Atributo de instância
6          self.idade = idade
7          # Atributo de instância
8
9      def emitir_som(self):
10         print("Som do animal")
11
12  # Criando Objetos
13  # Criando instâncias (objetos)
14  # da classe Animal
15  animal1 = Animal("Leão", 5)
16  animal2 = Animal("Elefante", 10)
17
18  # Acessando atributos e métodos
19  print(animal1.nome)
20  # Leão
21  print(animal2.idade)
22  # 10
23  animal1.emitir_som()
24  # Som do animal
```



```
1  # Encapsulamento
2  class Animal:
3      def __init__(self, nome, idade):
4          self.__nome = nome
5          # Atributo privado
6          self.__idade = idade
7          # Atributo privado
8
9      def emitir_som(self):
10         print("Som do animal")
11
12         # Métodos públicos para
13         # acessar os atributos privados
14     def get_nome(self):
15         return self.__nome
16
17     def set_nome(self, nome):
18         self.__nome = nome
19
20 # Criando um objeto
21 animal = Animal("Tigre", 3)
22
23 # Acessando atributos privados
24 # através de métodos
25 print(animal.get_nome())
26 # Tigre
27 animal.set_nome("Leopardo")
28 print(animal.get_nome())
29 # Leopardo
```



```
1  # Herança
2  class Animal:
3      def __init__(self, nome, idade):
4          self.nome = nome
5          self.idade = idade
6
7      def emitir_som(self):
8          print("Som do animal")
9
10 # Subclasse
11 class Cachorro(Animal):
12     def __init__(self, nome, idade, raca):
13         super().__init__(nome, idade)
14         # Chama o construtor da superclasse
15         self.raca = raca
16
17     def emitir_som(self):
18         print("Latido")
19
20 # Criando um objeto da subclasse
21 cachorro = Cachorro("Rex", 2, "Labrador")
22 print(cachorro.nome) # Rex
23 print(cachorro.raca) # Labrador
24 cachorro.emitir_som() # Latido
```



```
1  # Poliformismo
2  class Animal:
3      def emitir_som(self):
4          print("Som do animal")
5
6  class Cachorro(Animal):
7      def emitir_som(self):
8          print("Latido")
9
10 class Gato(Animal):
11     def emitir_som(self):
12         print("Miau")
13
14 # Função que demonstra polimorfismo
15 def fazer_animal_emitir_som(animal):
16     animal.emitir_som()
17
18 # Criando objetos
19 cachorro = Cachorro()
20 gato = Gato()
21
22 # Usando polimorfismo
23 fazer_animal_emitir_som(cachorro)
24 # Latido
25 fazer_animal_emitir_som(gato)
26 # Miau
```

# 05

## TRATAMENTO DE EXCEÇÕES

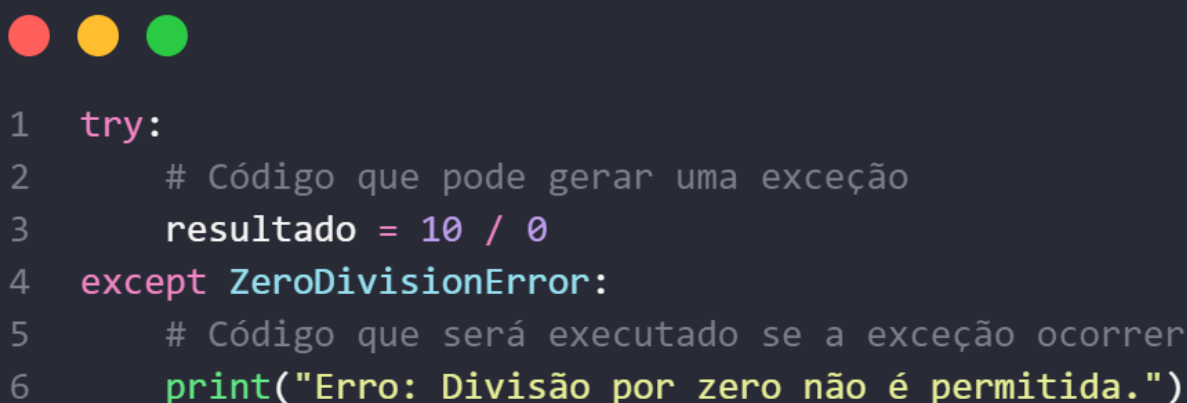
# Exceções em Python

Uma exceção é um evento que interrompe o fluxo normal de execução de um programa. Em Python, as exceções são objetos que representam erros e podem ser tratadas para evitar que o programa pare inesperadamente.

## Blocos 'try' e 'except'

Para tratar exceções, utilizamos os blocos 'try' e 'except'.

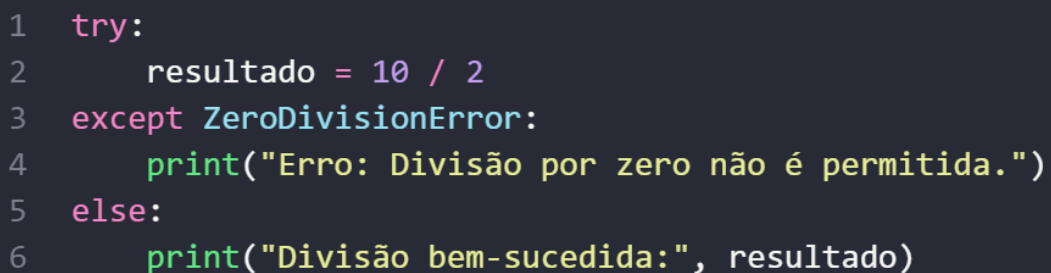
Exemplo:



```
1  try:
2      # Código que pode gerar uma exceção
3      resultado = 10 / 0
4  except ZeroDivisionError:
5      # Código que será executado se a exceção ocorrer
6      print("Erro: Divisão por zero não é permitida.")
```

## Bloco 'else'

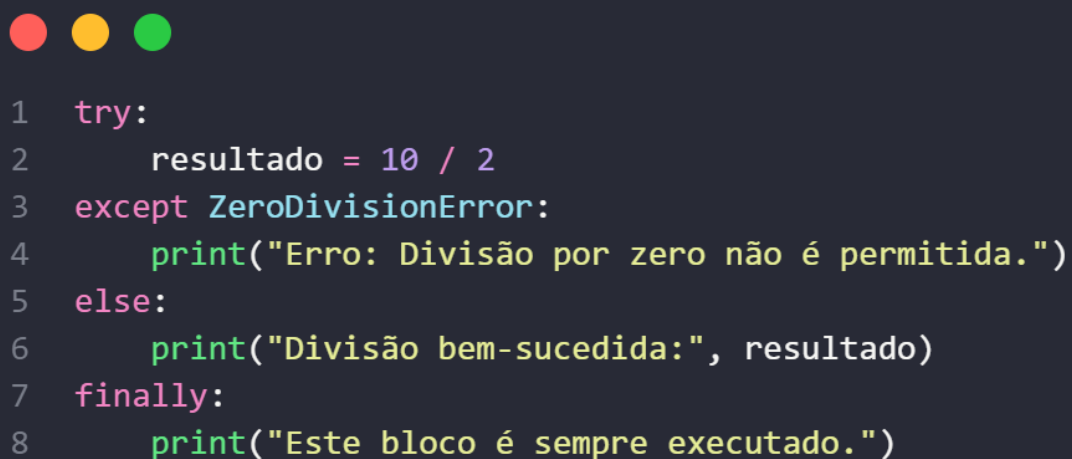
O bloco 'else' pode ser usado para executar código se nenhuma exceção for levantada.  
Exemplo:



```
1  try:
2      resultado = 10 / 2
3  except ZeroDivisionError:
4      print("Erro: Divisão por zero não é permitida.")
5  else:
6      print("Divisão bem-sucedida:", resultado)
```

## Bloco 'finally'

O bloco 'finally' é usado para executar código que deve ser executado independentemente de uma exceção ter sido levantada ou não. É útil para liberar recursos ou realizar limpeza.  
Exemplo:



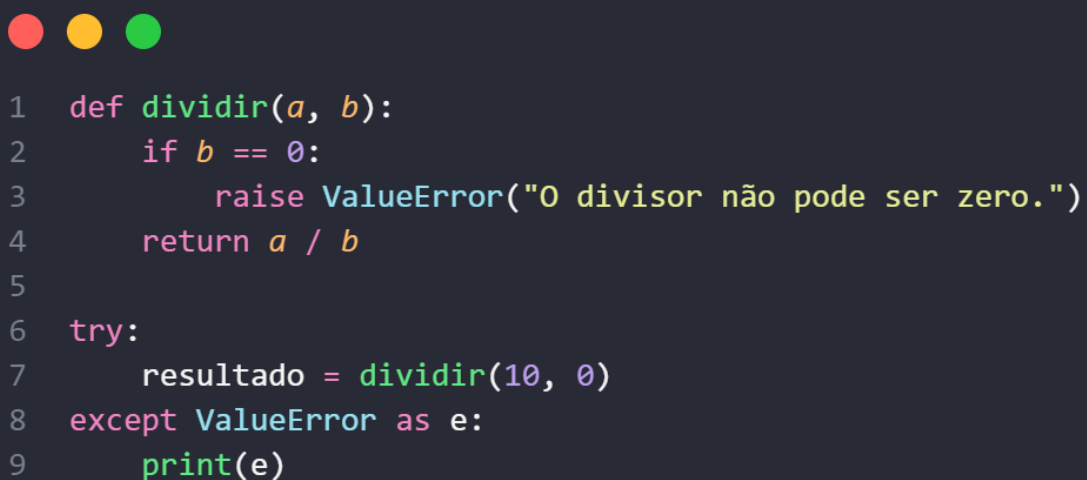
```

1  try:
2      resultado = 10 / 2
3  except ZeroDivisionError:
4      print("Erro: Divisão por zero não é permitida.")
5  else:
6      print("Divisão bem-sucedida:", resultado)
7  finally:
8      print("Este bloco é sempre executado.")

```

## Levantando Exceções

Você pode levantar suas próprias exceções usando a palavra-chave *'raise'*.  
Exemplo:



```

1  def dividir(a, b):
2      if b == 0:
3          raise ValueError("O divisor não pode ser zero.")
4      return a / b
5
6  try:
7      resultado = dividir(10, 0)
8  except ValueError as e:
9      print(e)

```



# Erros comuns em Python:

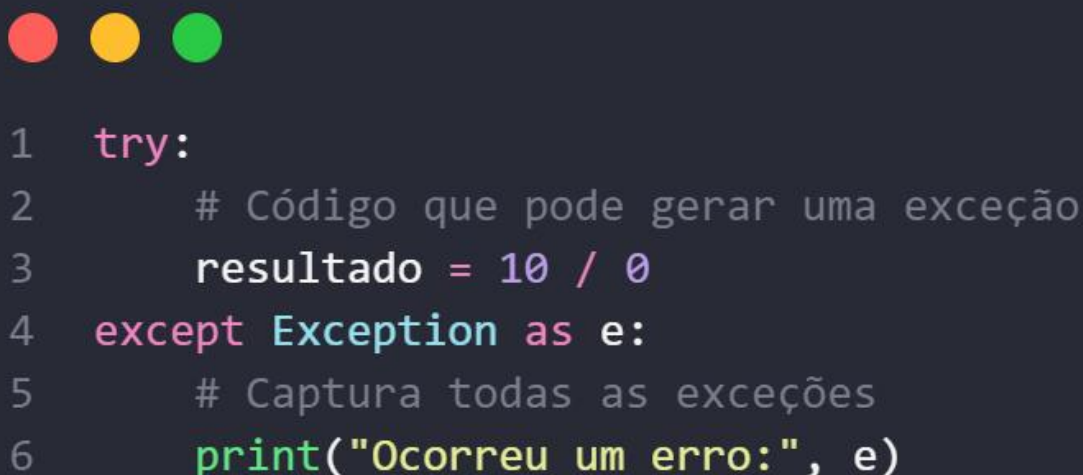


```
1  # SyntaxError
2  # Esquecendo dois pontos no final da definição de uma função
3  def minha_funcao()
4      print("Olá, Mundo!")
5
6  # NameError
7  # Usando uma variável que não foi definida
8  print(x)
9
10 # TypeError
11 # Tentando concatenar uma string com um número
12 resultado = "Olá" + 123
13
14 # IndexError
15 # Acessando um índice inexistente em uma lista
16 lista = [1, 2, 3]
17 print(lista[5])
18
19 # KeyError
20 # Acessando uma chave inexistente em um dicionário
21 dicionario = {"nome": "João"}
22 print(dicionario["idade"])
23
24 # AttributeError
25 # Tentando acessar um método inexistente em uma string
26 texto = "Olá, Mundo!"
27 texto.append("!!!")
28
29 # ValueError
30 # Convertendo uma string não numérica para um inteiro
31 numero = int("abc")
32
33 # ImportError
34 # Tentando importar um módulo inexistente
35 import modulo_inexistente
```

## Tratamento de Exceções Genéricas

Embora seja uma prática recomendada tratar exceções específicas, você pode usar uma exceção genérica para capturar todos os tipos de exceções.

Exemplo:



```
1  try:
2      # Código que pode gerar uma exceção
3      resultado = 10 / 0
4  except Exception as e:
5      # Captura todas as exceções
6      print("Ocorreu um erro:", e)
```

Conhecer os erros comuns em Python e como tratá-los pode ajudar a evitar problemas durante a execução do programa e a fornecer uma experiência mais suave e previsível para os usuários.

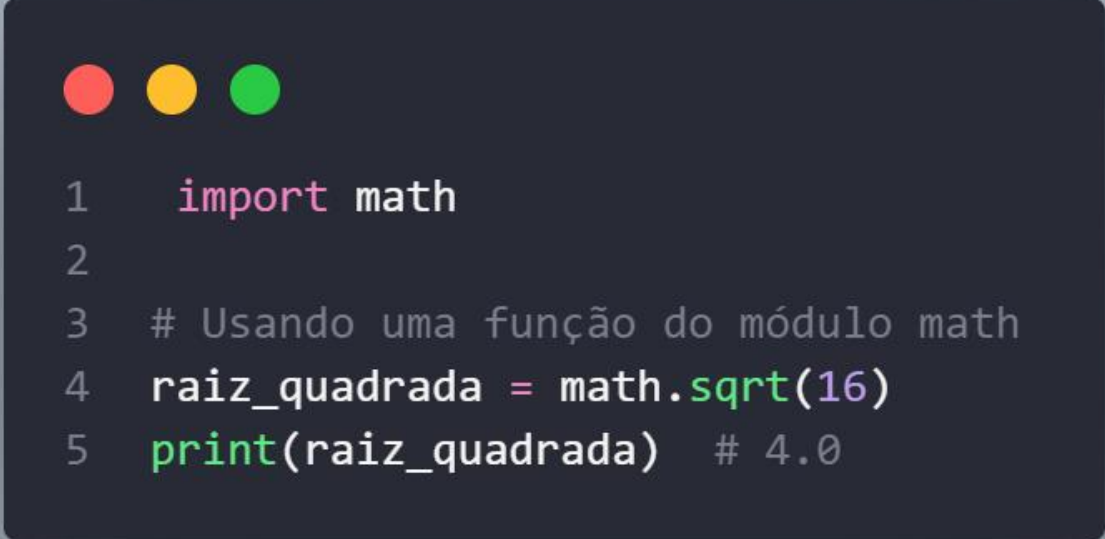
# 06

## MÓDULOS E PACOTES

## Importando Módulos

Para importar um módulo em Python, usamos a palavra-chave *'import'* seguida do nome do módulo. Uma vez importado, você pode acessar as funções, classes e variáveis definidas no módulo usando a notação de ponto.

Exemplo Básico de Importação:

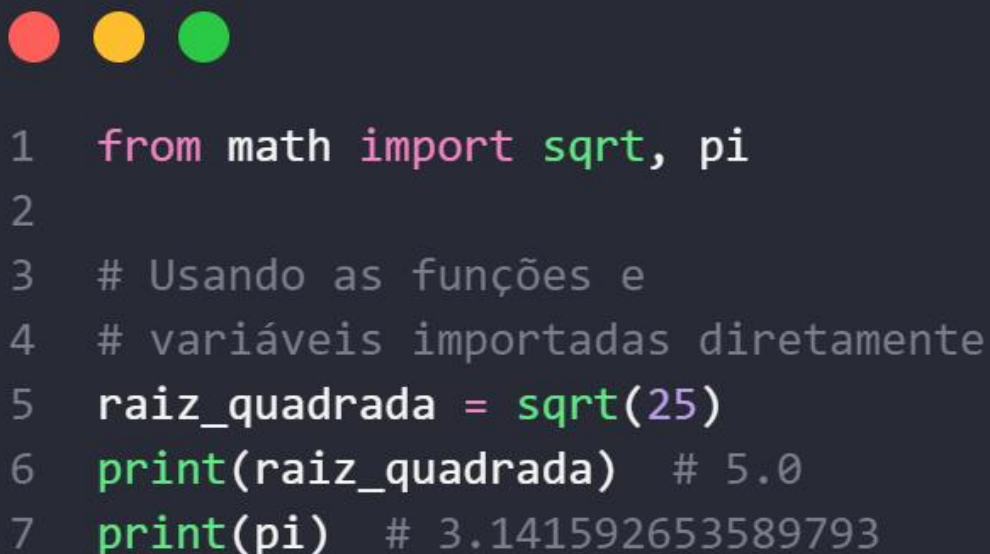
A screenshot of a code editor window with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. The code is written in Python and demonstrates importing the 'math' module and using the 'sqrt' function to calculate the square root of 16.

```
1  import math
2
3  # Usando uma função do módulo math
4  raiz_quadrada = math.sqrt(16)
5  print(raiz_quadrada)  # 4.0
```

# Importação Específica de Funções ou Classes

Você pode importar funções ou classes específicas de um módulo usando a sintaxe *'from module import name'*.

Exemplo:



```
1 from math import sqrt, pi
2
3 # Usando as funções e
4 # variáveis importadas diretamente
5 raiz_quadrada = sqrt(25)
6 print(raiz_quadrada) # 5.0
7 print(pi) # 3.141592653589793
```




## Renomeando Módulos e Funções

Você pode renomear um módulo ou uma função durante a importação usando a palavra-chave `as`. Isso é útil para simplificar nomes longos ou resolver conflitos de nomes.

```
1  import numpy as np
2
3  # Usando o módulo renomeado
4  array = np.array([1, 2, 3])
5  print(array)
6
7  from math import sqrt as raiz
8
9  # Usando a função renomeada
10 raiz_quadrada = raiz(36)
11 print(raiz_quadrada) # 6.0
```

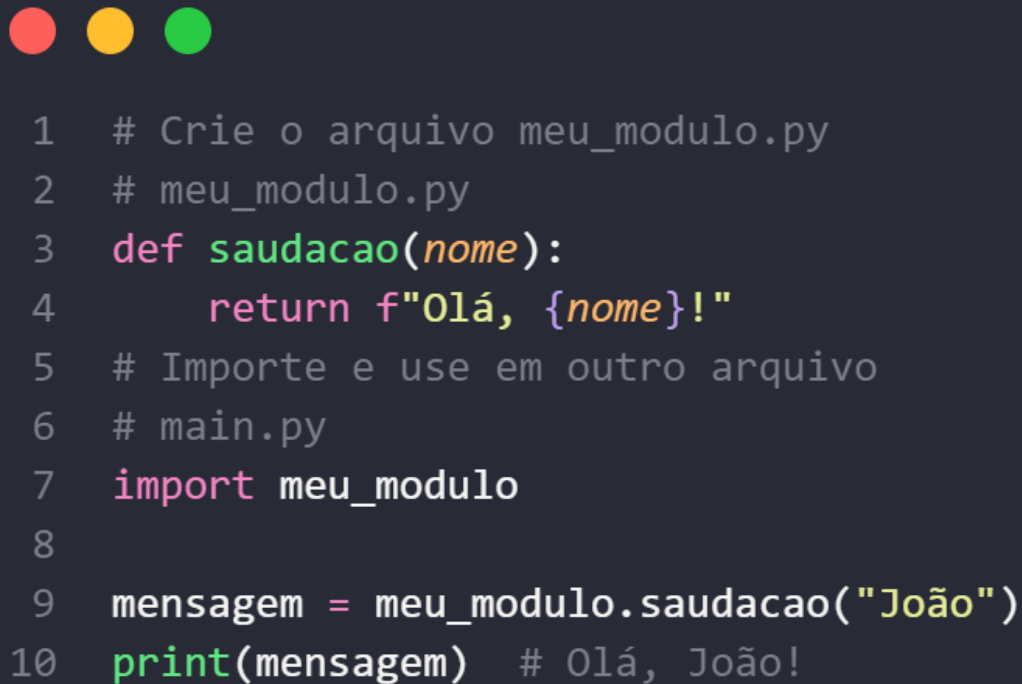
## Importando Todos os Itens de um Módulo

Para importar todos os itens de um módulo, use a sintaxe `from module import *`. No entanto, essa prática não é recomendada porque pode causar conflitos de nomes e tornar o código menos legível.



```
1  from math import *
2
3  # Usando funções do módulo math
4  raiz_quadrada = sqrt(49)
5  print(raiz_quadrada)  # 7.0
6  print(pi)  # 3.141592653589793
```

# Criando e importando seus próprios módulos



```
1  # Crie o arquivo meu_modulo.py
2  # meu_modulo.py
3  def saudacao(nome):
4      return f"Olá, {nome}!"
5  # Importe e use em outro arquivo
6  # main.py
7  import meu_modulo
8
9  mensagem = meu_modulo.saudacao("João")
10 print(mensagem)  # Olá, João!
```

## Importando Pacotes

Pacotes são coleções de módulos organizados em diretórios com um arquivo `__init__.py`. O arquivo `__init__.py` pode estar vazio ou pode inicializar o pacote.



```

1  # meu_pacote/__init__.py
2  # Este arquivo pode estar
3  # vazio ou inicializar o pacote
4
5  # meu_pacote/modulo1.py
6  def funcao_modulo1():
7      return "Função do módulo 1"
8
9  # Importando o módulo de um pacote
10 from meu_pacote import modulo1
11
12 resultado = modulo1.funcao_modulo1()
13 print(resultado)
14 # Função do módulo 1

```

## Bibliotecas Padrão e Externas

### Bibliotecas Padrão

Python vem com uma biblioteca padrão rica que cobre diversas áreas, como manipulação de arquivos, operações matemáticas, redes e mais. Você pode importar esses módulos diretamente sem instalação adicional.



```
1 import os
2
3 # Usando o módulo os para listar arquivos em um diretório
4 arquivos = os.listdir(".")
5 print(arquivos)
```

## Bibliotecas Externas

Para usar bibliotecas externas, você precisa instalá-las usando um gerenciador de pacotes como o *'pip'*. Depois de instaladas, você pode importá-las da mesma forma que importa módulos da biblioteca padrão.

Exemplo de Instalação e Importação de Biblioteca Externa:

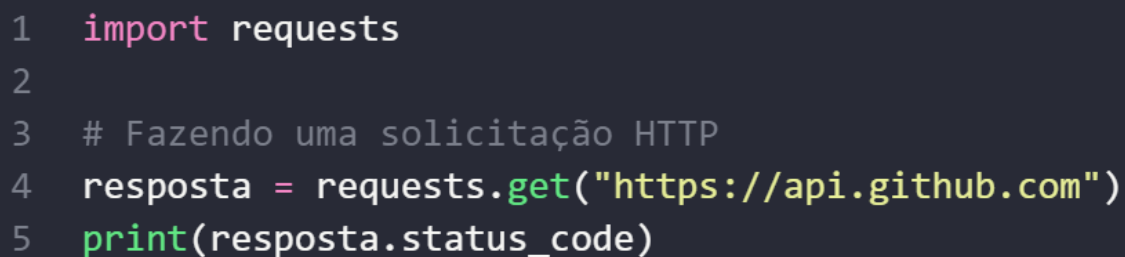
Instale a biblioteca usando *'pip'*:



```
1 pip install requests
```



## Importe e use a biblioteca:



```
1  import requests
2
3  # Fazendo uma solicitação HTTP
4  resposta = requests.get("https://api.github.com")
5  print(resposta.status_code)
```

A importação de módulos é uma parte essencial da programação em Python, permitindo que você organize seu código em componentes reutilizáveis e aproveite uma vasta gama de funcionalidades prontas.

Seja usando a biblioteca padrão, bibliotecas externas ou seus próprios módulos, a importação de módulos torna seu código mais modular, legível e fácil de manter.

## REFERÊNCIAS:

### 1. Python e sua História

- Python Software Foundation: (<https://www.python.org/doc/essays/history/>)
- Guido van Rossum's Blog: (<http://python-history.blogspot.com/>)

### 2. Instalação do Python e Ambiente de Desenvolvimento

- Python Software Foundation: (<https://www.python.org/downloads/>)
- Real Python: (<https://realpython.com/python-development-environments/>)

### 3. Variáveis, Tipos de Dados e Operadores

- Python Documentation: (<https://docs.python.org/3/library/stdtypes.html>)
- Real Python: (<https://realpython.com/python-data-types/>)

### 4. Estruturas de Controle: Condicionais e Loops

- Python Documentation: (<https://docs.python.org/3/tutorial/controlflow.html>)

- 
- Real Python: (<https://realpython.com/python-conditional-statements/>)

## 5. Funções e Modularidade

- Python Documentation: (<https://docs.python.org/3/tutorial/controlflow.html#defining-functions>)
- Real Python: (<https://realpython.com/defining-your-own-python-function/>)

## 6. Listas, Tuplas e Conjuntos

- Python Documentation: (<https://docs.python.org/3/library/stdtypes.html#sequence-types-list-tuple-range>)
- Real Python: (<https://realpython.com/python-lists-tuples/>)

## 7. Manipulação de Strings

- Python Documentation: (<https://docs.python.org/3/library/stdtypes.html#text-sequence-type-str>)
- Real Python: (<https://realpython.com/python-strings/>)

## 8. Conceitos Básicos de POO

- Python Documentation:  
(<https://docs.python.org/3/tutorial/classes.html>)
- Real Python:  
(<https://realpython.com/python3-object-oriented-programming/>)

## 9. Classes e Objetos em Python

- Python Documentation:  
(<https://docs.python.org/3/tutorial/classes.html#class-objects>)
- Real Python:  
(<https://realpython.com/python3-object-oriented-programming/>)

## 10. Exceções e Erros Comuns

- Python Documentation:  
(<https://docs.python.org/3/tutorial/errors.html>)
- Real Python: (<https://realpython.com/python-exceptions/>)

## 11. Importação de Módulos

- Python Documentation:  
(<https://docs.python.org/3/reference/import.html>)

- 
- Real Python: (<https://realpython.com/python-modules-packages/>)