

Programa de Formação em Python

Aula 1: Introdução ao Python e Configuração do Ambiente

Objetivo: Familiarizar os alunos com Python e o ambiente de desenvolvimento.

- Instalação do Python e IDEs (VS Code, PyCharm, etc.).
- Conceitos básicos de programação: o que é programação, scripts e execução de código.
- Primeiros passos com Python: sintaxe básica e comandos simples.
- Operadores e variáveis: tipos de dados (inteiros, floats, strings).
- Exercícios simples de entrada e saída.

Aula 2: Controle de Fluxo

Objetivo: Aprender estruturas de controle de fluxo.

- Estruturas condicionais: if, else, elif.
- Estruturas de repetição: for e while.
- Compreensão sobre loops aninhados.
- Introdução às funções embutidas.
- Exercícios práticos envolvendo controle de fluxo.

Aula 3: Funções e Estruturas de Dados Básicas

Objetivo: Introduzir funções e manipulação de listas.

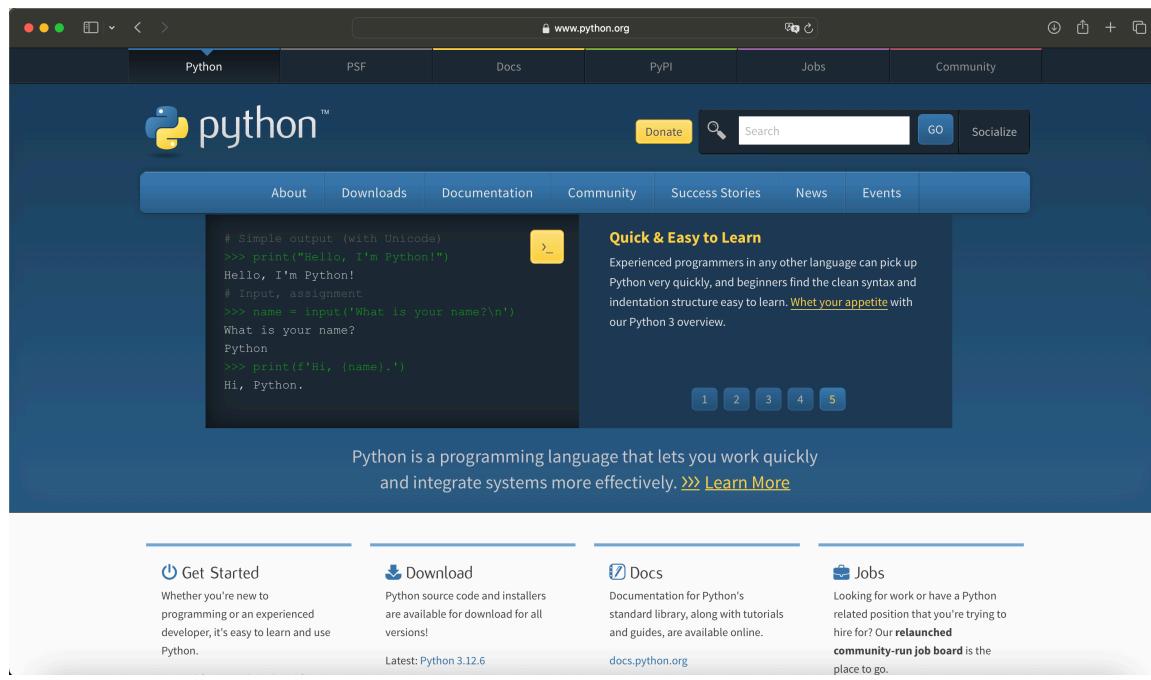
- Definição e chamada de funções.
- Parâmetros e retorno de valores.
- Introdução às listas: criação, indexação e manipulação.
- Métodos comuns de listas (append, remove, sort, etc.).
- Exercícios práticos envolvendo funções e listas.

1 - Introdução ao Python e Configuração do Ambiente

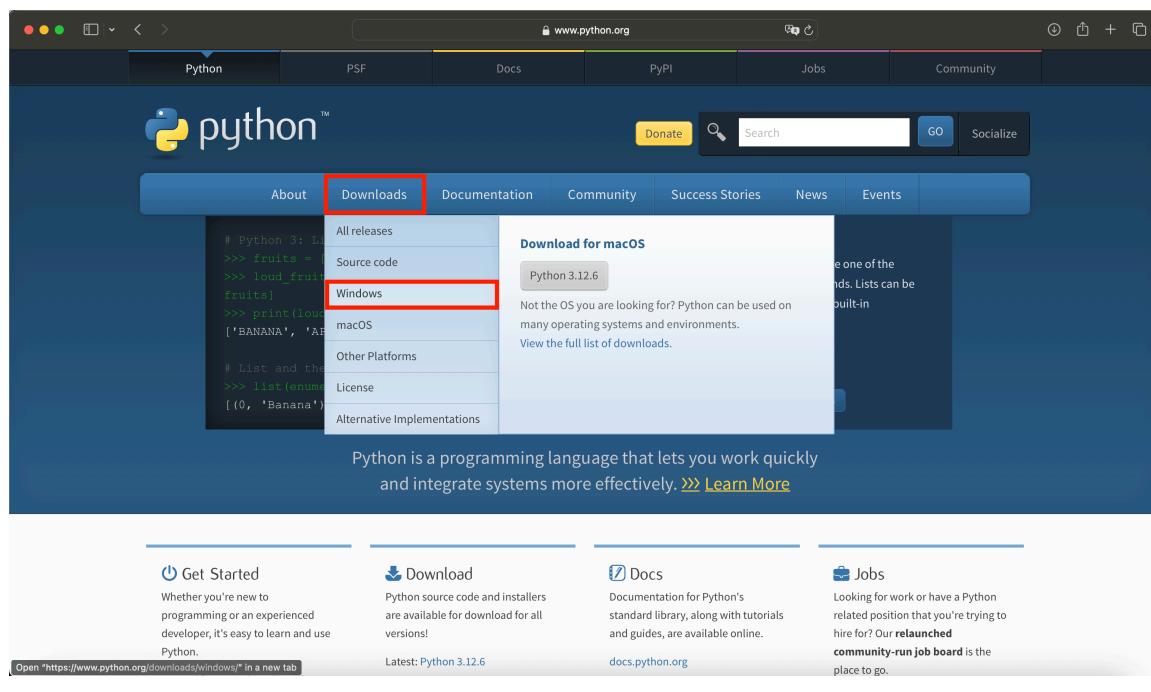
1.1 - Instalação do Python e Pycharm

Vamos começar fazendo download e instalação do python

Passo 1: Abra o browser (navegador) da sua preferência e entre no site [python.org](https://www.python.org). aparecerá a seguinte pagina:



Passo 2: Sobreponha o cursor sobre Downloads e a seguir escolha a plataforma que você está a usar. No exemplo a escolha foi windows.



Passo 3: Escolha uma versão estável que tenha arquivos para download, escolha um arquivo e clique sobre ele e aguarde o término do download. Obs: escolha o arquivo baseado em sua plataforma 64-bit ou 32-bit

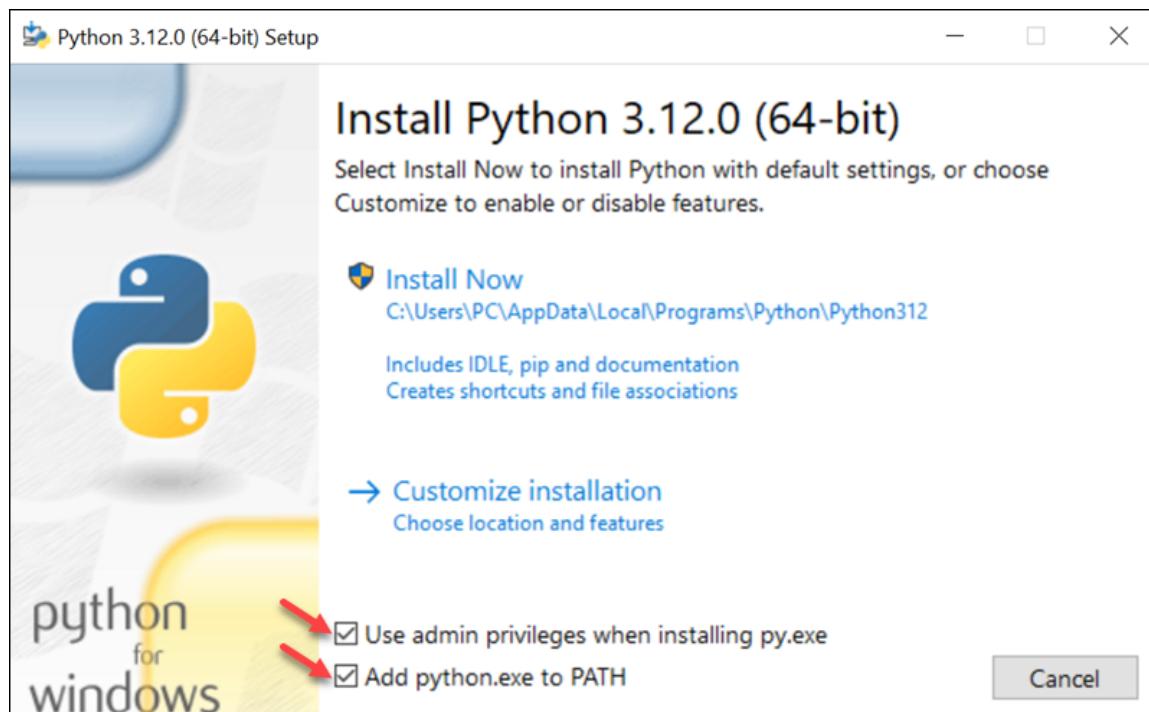
The screenshot shows the Python Releases for Windows page. The 'Downloads' section is active. It lists stable releases (3.11.10, 3.10.15, 3.12.6) and pre-releases (3.13.0rc2). For each release, it provides links to download Windows installers for 64-bit and 32-bit architectures, as well as ARM64 and embeddable packages. The 'Download Windows installer (64-bit)' link for Python 3.12.6 is highlighted with a red box.

Passo 4: Execute o arquivo baixado.

Passo 5: A janela de instalação vai mostrar duas caixas de verificação:

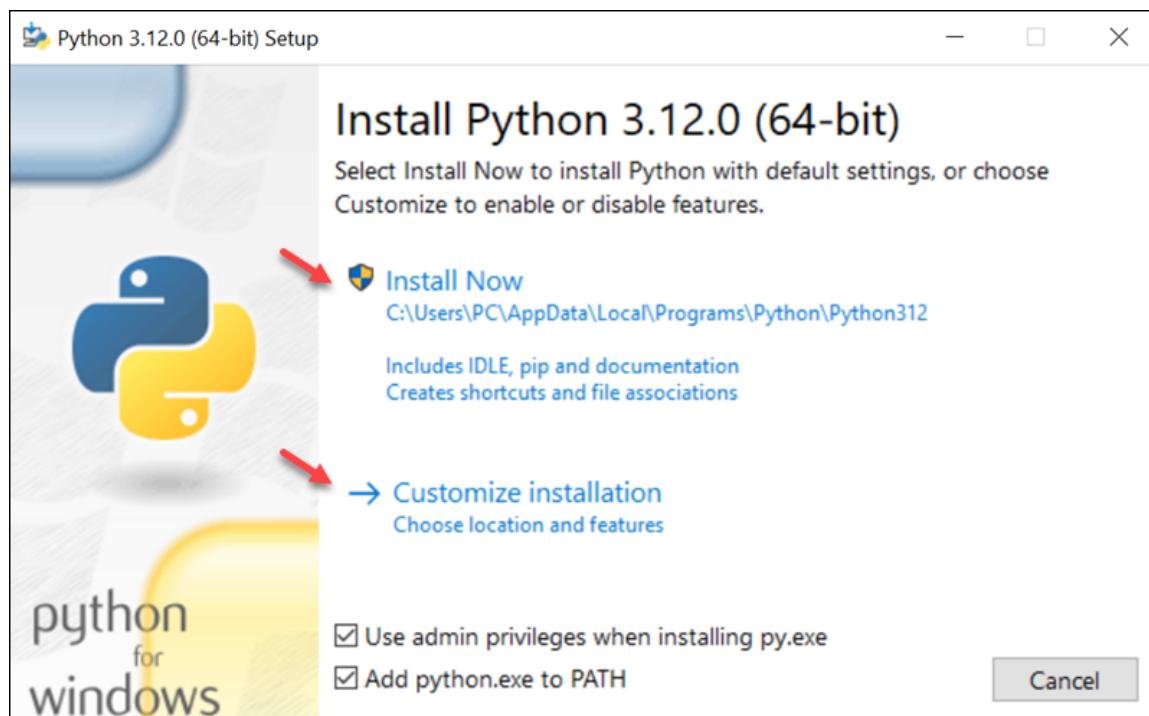
- **Admin Privileges:** O parâmetro controla se o Python deve ser instalado para o utilizador atual ou para todos os utilizadores do sistema. Essa opção permite que você altere a pasta de instalação do Python.
- **Add Python to Path:** A segunda opção coloca o executável na variável PATH após a instalação. Também pode adicionar Python à variável de ambiente PATH manualmente mais tarde.

Para uma instalação mais simples, recomendo que assinale ambas as caixas de verificação.

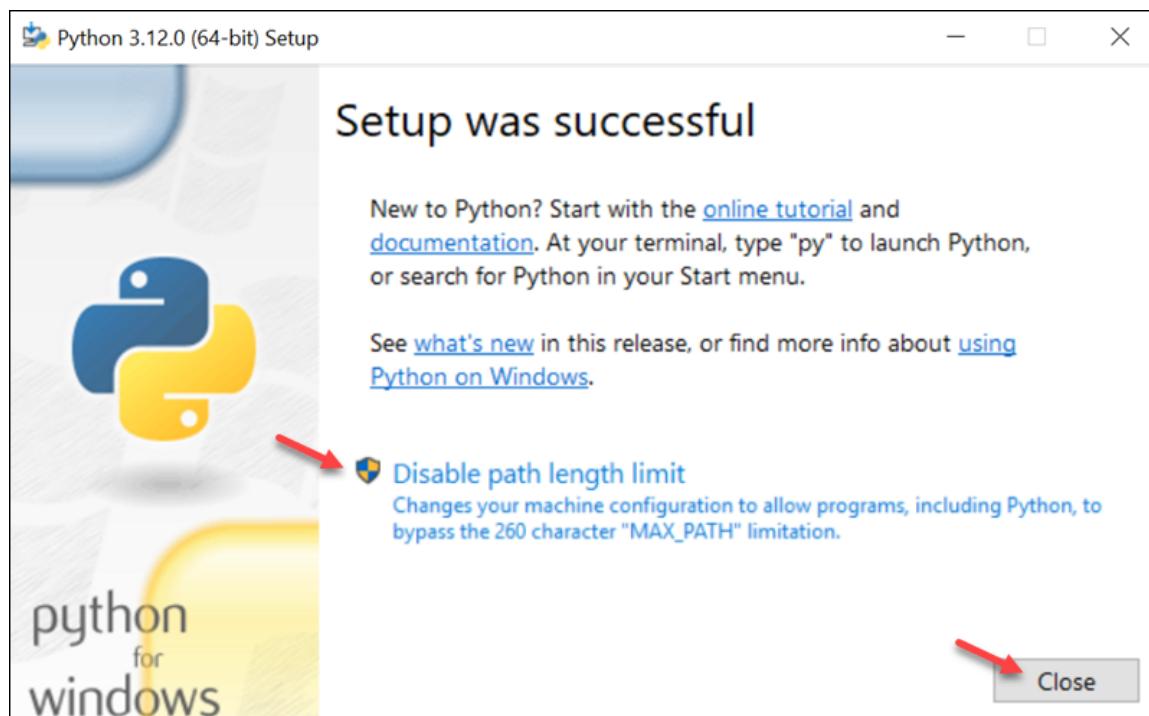


Passo 6: Selecione a opção Instalar agora para a instalação recomendada (nesse caso, ignore os dois passos seguintes).

Para ajustar as opções de instalação predefinidas, selecione Personalizar instalação e avance para o passo seguinte.



Passo 7: Selecione se pretende desativar o limite de comprimento do caminho. Escolher esta opção irá permitir ao Python contornar o limite MAX_PATH de 260 caracteres.



Passo 8: Verificar se o Python foi instalado no Windows

A primeira maneira de verificar se o Python foi instalado com sucesso é através da linha de comando. Abra o prompt de comando e execute o seguinte comando:

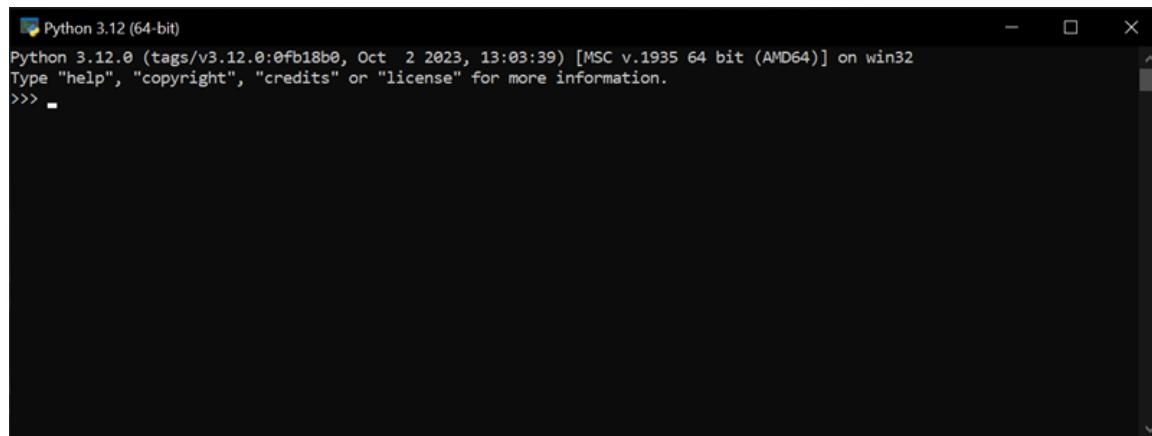
```
> python --version
```

```
C:\Users\PC>python --version
Python 3.12.0
```

O resultado mostra a versão do Python instalada.

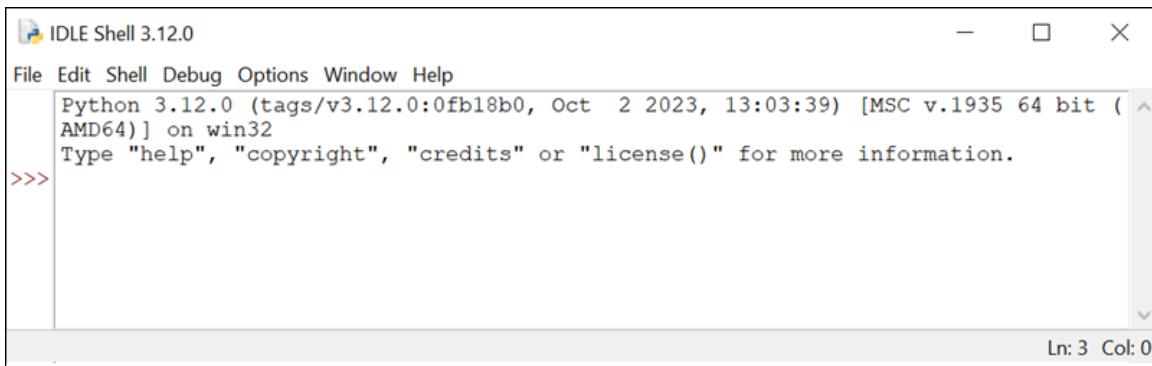
A segunda forma é utilizar a GUI (Graphical User Interface) para verificar a instalação do Python. Siga os passos abaixo para executar o interpretador Python ou IDLE (Integrated Development and Learning Environment):

1. Navegue até ao diretório onde o Python foi instalado no sistema.
2. Faça duplo clique em python.exe (o interpretador Python) ou IDLE.
3. O interpretador abre a linha de comandos e mostra a seguinte janela:



The screenshot shows a terminal window titled "Python 3.12 (64-bit)". The window displays the following text:
Python 3.12.0 (tags/v3.12.0:0fb18b0, Oct 2 2023, 13:03:39) [MSC v.1935 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>

Executando o IDLE abre o IDE incorporado do Python:

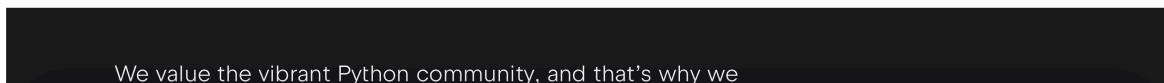


Em ambos os casos, a versão instalada do Python aparece no ecrã e o editor está pronto a ser utilizado.

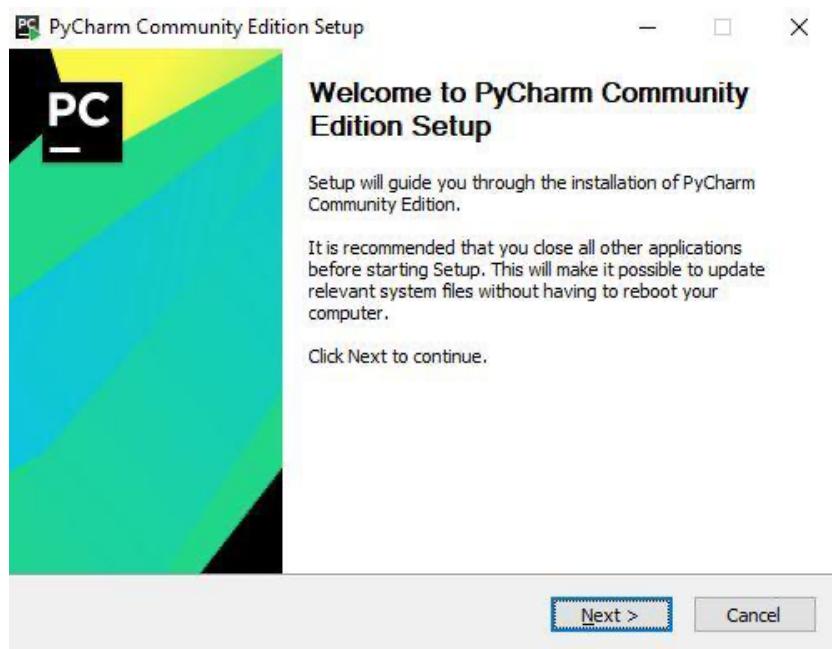
Com o python instalado vamos fazer o download e instalação do pycharm:

Passo 1: Aceda ao site oficial da Jetbrains <https://www.jetbrains.com/pycharm/download/?section=windows> e clique no link “DOWNLOAD” da secção Comunidade.

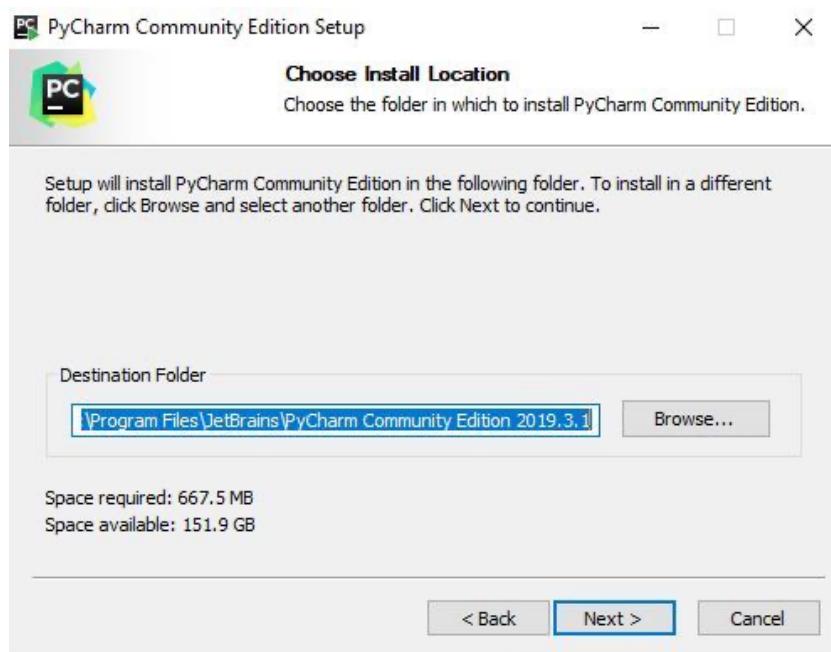
A screenshot of the PyCharm Professional download page on the JetBrains website. The top navigation bar includes links for Developer Tools, Team Tools, Education, Solutions, Support, Store, and a search bar. The main content area features the PyCharm logo and the text "The Python IDE for data science and web development". It shows a preview of the PyCharm interface with a project named "django-tutorial-extended". Below the interface preview, there are download buttons for "Download" and ".dmg (Intel)" (with a note about selecting an installer for Intel or Apple Silicon). A "Free 30-day trial" button is also present. At the bottom, there is a footer with links for "Version: 2024.2.2", "System requirements", "Other versions", "Installation instructions", and "Third-party software".



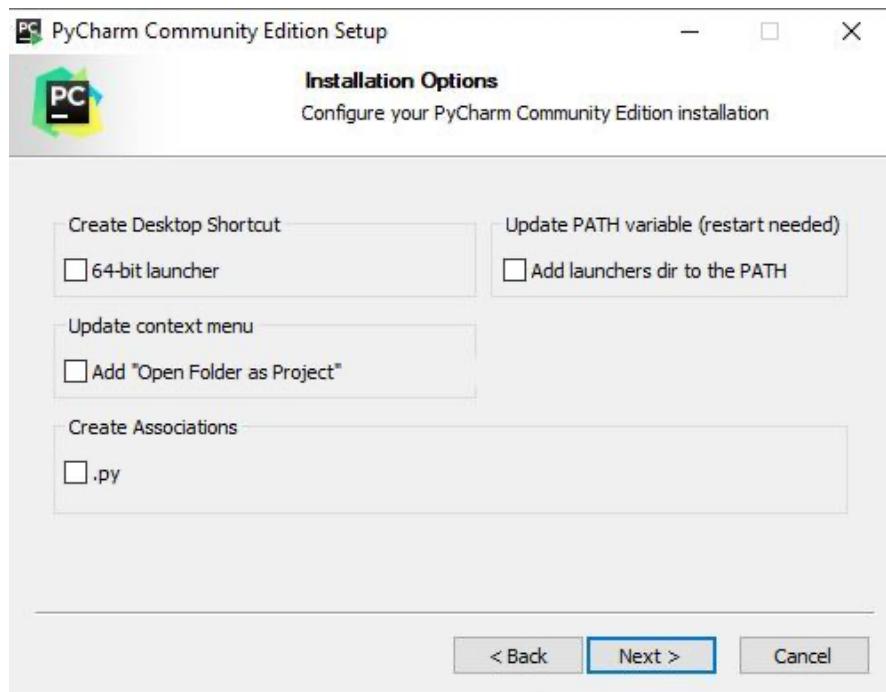
Passo 2: Com o download terminado execute o arquivo baixado, abrirá uma janela e nela click no botão “Next” ou “Seguinte” ou “Próximo” como forme a configuração de linguagem do seu computador.



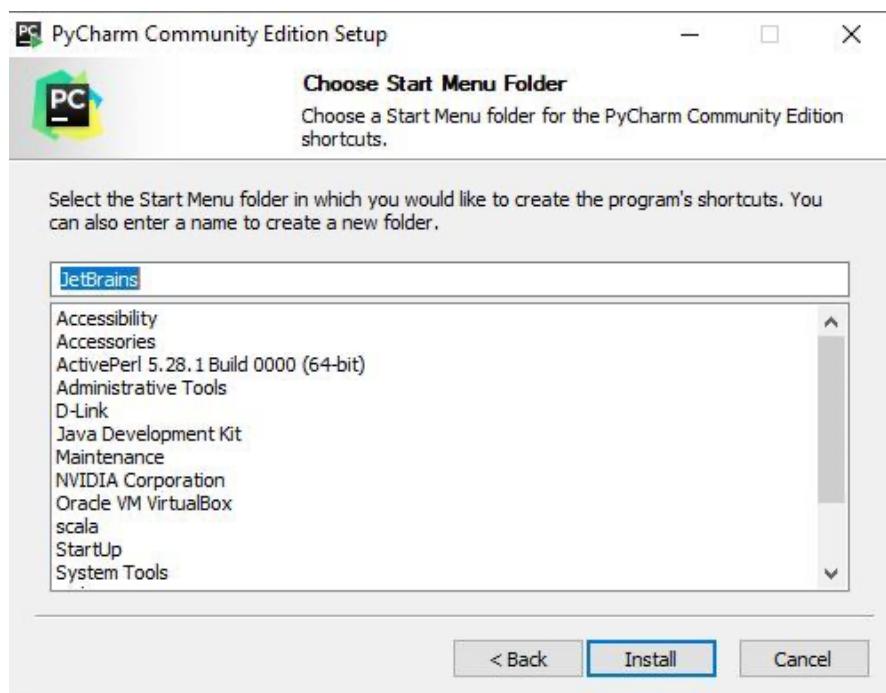
Passo 3: Depois de clicar em Next (Seguinte/Próximo), tem de escolher a pasta de destino de acordo com a sua opção e click em Next (Seguinte/Próximo)



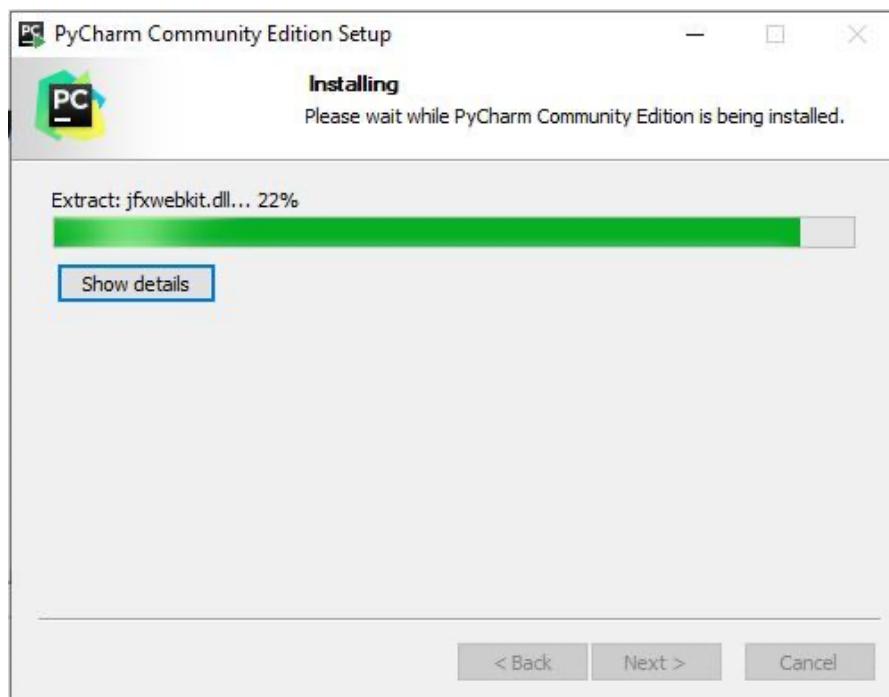
Passo 4: Escolha as opções de instalação de acordo com a sua escolha e click em Next (Seguinte/Próximo). Recomendo selecionar as três primeiras.



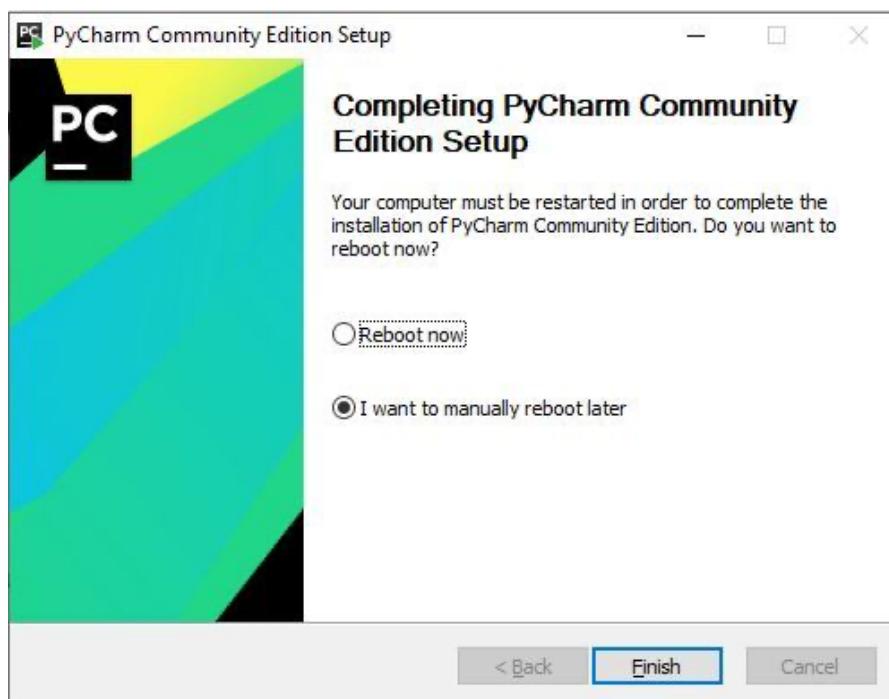
Passo 5: Escolha JetBrains e click em Instalar.



Passo 6: Deixe a instalação terminar.



Passo 7: Depois de concluir a instalação, será mostrado que o Pycharm foi instalado com sucesso e, em seguida, clique em “I want to manually reboot later” (Quero reiniciar manualmente mais tarde). Clique em Finish (Concluir) e o processo estará concluído.



1.2 - Conceitos Básicos de Programação

1.2.1 - O que é Programação?

Programação é o processo de criar e desenvolver instruções que permitem a um computador executar tarefas específicas. Essas instruções são escritas em uma linguagem de programação, que serve como meio de comunicação entre o programador e o computador.

Resumidamente, a programação envolve a criação de um código que define o comportamento de um software, desde tarefas simples, como somar dois números, até complexos sistemas automatizados.

Principais conceitos:

1. Código: As instruções escritas pelo programador para o computador.
2. Linguagem de Programação: O conjunto de regras e sintaxe que define como o código deve ser escrito (ex: Python, Java, C++, etc.).
3. Algoritmo: Um passo a passo detalhado para resolver um problema ou realizar uma tarefa.
4. Compilação/Interpretação: O processo pelo qual o código é traduzido para uma linguagem que o computador entende (código de máquina).
5. Execução: O momento em que o computador segue as instruções do código e realiza as ações programadas.

1.2.2 - O que é um Script?

Um script é um tipo de programa simples escrito em uma linguagem de script (como Python, JavaScript, Bash, etc.) que é executado diretamente sem a necessidade de compilação prévia. Ele é utilizado para automatizar tarefas repetitivas, realizar operações específicas ou configurar ambientes, geralmente de forma rápida e fácil.

Características de um script:

1. Interpretação: Scripts são normalmente interpretados linha a linha em vez de compilados, o que significa que são executados diretamente por um interpretador.
2. Automação: Comumente usados para automatizar tarefas rotineiras, como mover arquivos, processar dados ou configurar ambientes.
3. Facilidade de Uso: Não há processo complexo de compilação, tornando-os rápidos para modificar e testar.
4. Flexibilidade: São frequentemente usados para pequenas tarefas, mas também podem servir para desenvolvimento mais complexo.

Exemplo em Python:

```
Python Console
>>> print("Olá Mundo")
Olá Mundo
>>> |
```

Esse simples script imprime a mensagem “Olá mundo” no terminal quando executado.

1.2.3 - O que é Execução de Código?

Execução de código é o processo pelo qual um computador segue as instruções escritas por um programador em um programa ou script. Essas instruções são processadas sequencialmente, levando o computador a realizar tarefas como cálculos, manipulação de arquivos ou exibição de resultados.

Etapas da execução de código:

1. Escrita do código: O programador escreve o código em uma linguagem de programação.
2. Interpretação ou Compilação:
 - Compilação: O código é convertido para código de máquina (linguagens compiladas como C, Java).
 - Interpretação: O código é executado linha por linha por um interpretador (linguagens como Python, JavaScript).
3. Execução: O computador segue as instruções fornecidas, realizando operações como cálculos e manipulação de dados.
4. Saída: O resultado do código, seja visível (ex: texto na tela) ou invisível (ex: alteração de um arquivo).

Exemplo em Python:

```
Python Console
>>> a = 5
>>> b = 3
>>> print(a + b)
8
>>> |
```

Este código realiza a soma de dois números e exibe o resultado (8) no console.

1.2.4 Execução de código

É o processo pelo qual um compilador ou interpretador de linguagem de programação realiza as instruções definidas em um programa ou script. Em termos simples, quando você escreve um código, ele é uma série de comandos ou instruções. A execução é o ato de o computador seguir essas instruções e realizar as tarefas solicitadas, como cálculos, exibição de resultados na tela, manipulação de dados, entre outros.

Como funciona a execução de código em Python:

No caso do **Python**, que é uma linguagem interpretada, o código é executado linha por linha por um interpretador Python. O interpretador lê o código-fonte, converte-o em instruções comprehensíveis pelo computador e executa essas instruções imediatamente.

Exemplo de execução de código:

Imagine o seguinte código simples em Python:

```
Python Console
>>> print("Olá Mundo")
Olá Mundo
>>> |
```

Quando você executa esse código:

1. O interpretador Python lê a instrução `print("Olá, mundo!")`.
2. Ele entende que essa instrução solicita a exibição da frase "Olá, mundo!" na tela.
3. O interpretador executa a instrução e imprime o resultado na tela.

Tipos de execução:

- **Execução de código em tempo de compilação:** Ocorre em linguagens compiladas, como C ou Java, onde o código-fonte é convertido em código de máquina antes da execução. O programa é "compilado" primeiro e depois executado.
- **Execução de código em tempo de execução:** Como em Python, o código é interpretado e executado diretamente sem a necessidade de compilar antes, o que torna o processo mais rápido e interativo para o programador.

A execução é o momento em que o código sai da fase de escrita e passa a ser "entendido" pela máquina, transformando as instruções em ações reais.

1.3 Primeiros passos com Python

1.3.1 sintaxe básica

A **sintaxe** de uma linguagem de programação se refere ao conjunto de regras que determinam como você deve escrever o código para que ele seja compreendido pelo interpretador ou compilador. Python é conhecido por sua **simplicidade** e legibilidade, sendo uma das linguagens mais fáceis para iniciantes.

1. Indentação

Indentação é o **espaçamento** que se coloca no início de uma linha de código para organizar e estruturar os blocos de um programa

Em Python, a **indentação** é usada para delimitar blocos de código. Diferente de outras linguagens como C ou Java, onde você usa chaves {} para definir blocos, em Python, o espaçoamento (geralmente 4 espaços ou um "tab") é o que define os blocos.

Exemplo:

```
if 5 > 2:  
    print("5 é maior que 2") # Indentação correta
```

2. Comentário

Comentário é uma parte do código que é ignorada pelo compilador ou interpretador e, portanto, não é executada. Seu propósito é fornecer explicações, anotações ou observações para quem estiver lendo o código, seja o próprio programador ou outras pessoas que vão trabalhar com o mesmo projeto. Comentários são usados para tornar o código mais claro, descrever a funcionalidade de trechos específicos ou deixar observações sobre mudanças e futuras melhorias.

Funções de um Comentário:

```
# Calcula a soma de dois números  
soma = num1 + num2
```

Explicação do Código: Comentários podem ser usados para descrever o que um trecho de código faz, especialmente quando o código é complexo ou não é imediatamente claro.

Documentação: Podem documentar o comportamento de funções, classes ou módulos.

```
def calcular_area(base, altura):
    """
    Função que calcula a área de um triângulo.
    :param base: valor da base do triângulo
    :param altura: valor da altura do triângulo
    :return: área do triângulo
    """
    return (base * altura) / 2
```

Desativar Temporariamente Trechos de Código: Comentários podem ser usados para "comentar" partes do código que você não deseja executar temporariamente, como durante testes ou depuração.

```
# print("Esta linha não será executada")
print("Esta linha será executada")
```

Tipos de Comentários:

Comentário de Linha Única: Em Python, usa-se o símbolo # para criar um comentário de linha única. Tudo o que estiver depois de # na mesma linha será ignorado.

```
# Isto é um comentário de linha única
print("Olá, mundo!") # Este também é um comentário
```

Comentário de Múltiplas Linhas: Em algumas linguagens, como C e Java, existem símbolos específicos para comentários de várias linhas. Em Python, embora não haja um símbolo específico para isso, você pode usar múltiplos # ou uma string de múltiplas linhas (geralmente entre aspas triplas """ ou ''') para criar um comentário em bloco.

Exemplo usando #:

```
# Este é um comentário de múltiplas linhas
# em Python, usando o símbolo # em
# cada linha.
```

Exemplo usando strings múltiplas:

```
"""
Este é um comentário de múltiplas linhas
em Python, usando uma string entre aspas triplas.
Embora seja tecnicamente uma string,
ela não será executada, pois não está atribuída a nada.
"""
```

Função print(): A função print() é usada para exibir informações na tela. Você pode passar strings ou variáveis como argumento para exibi-las:

```
print("Olá, mundo!")
```

1.3.2 Comandos simples.

A seguir veremos alguns comandos simples que você pode usar logo nos primeiros passos com os python:

1. **Atribuição de variáveis**: é a operação que define que uma dada variável terá um dado valor, e esta operação é efectuada com o operador matemático de igualdade (=)

```
nome = "Ana"
idade = 25
print(nome, idade)
```

No exemplo acima atribuímos o valor “Ana” a variável nome e 25 a variável idade.

2. **Entrada de dados**: é a ação de inserir dados por meio de um dispositivo de entrada (teclado, mouse, etc) para o computador.

A função input() do python permite que o utilizador insira dados.

Exemplo:

```
nome = input("Qual é o seu nome? ")
print("Olá,", nome)
```

1.4 Operadores e Variáveis

1.4.1 Operadores

Operadores: operadores são símbolos que realizam operações sobre variáveis e valores. Python suporta vários tipos de operadores:

Operadores Aritméticos: Esses operadores realizam operações matemáticas básicas;

- `+`: Adição
- `-`: Subtração
- `*`: Multiplicação
- `/`: Divisão
- `**`: Exponenciação (potência)
- `%`: Módulo (resto da divisão)

Exemplo:

```
a = 10
b = 3
print(a + b) # 13
print(a - b) # 7
print(a * b) # 30
print(a / b) # 3.33 (divisão com float)
print(a ** b) # 1000 (10 elevado a 3)
print(a % b) # 1 (resto da divisão de 10 por 3)
```

Operadores de Comparaçāo: Esses operadores comparam dois valores e retornam um valor booleano (True ou False)

- `==`: Igual a
- `!=`: Diferente de
- `>`: Maior que
- `<`: Menor que
- `>=`: Maior ou igual a
- `<=`: Menor ou igual a

Exemplo:

```
x = 5
y = 10
print(x == y) # False
print(x != y) # True
print(x < y) # True
```

Operadores Lógicos: Operadores lógicos são usados para combinar expressões que resultam em um valor lógico;

- and: Retorna True se ambas as expressões forem verdadeiras
- or: Retorna True se pelo menos uma expressão for verdadeira
- not: Inverte o valor lógico (True se for False, e vice-versa)

Exemplo:

```
x = True
y = False
print(x and y)  # False
print(x or y)   # True
print(not x)    # False
```

1.4.2 Variáveis e Tipos de dados (inteiros, floats, strings)

Variáveis são um identificadores de memória que servem para armazenar dados, que podem ser de diferentes tipos: (int, float, String, boolean, etc).

Os tipos de dados básicos em Python incluem:

Strings: Uma string é uma sequência de caracteres. Em Python, elas são representadas por aspas simples ou duplas;

```
nome = "Python"
print(nome)
```

Números Inteiros (int): são números sem partes decimais;

```
idade = 30
print(idade)
```

Números de Ponto Flutuante (float): são números com casas decimais;

```
altura = 1.75
print(altura)
```

1.5 Exercícios simples de entrada e saída

Exercício 1: O que é indentação e por que ela é importante em Python?

Exercício 2: Explique a diferença entre um comentário de linha única e um comentário de múltiplas linhas em Python.

Exercício 3: O que acontece se você esquecer de usar a indentação correta em um bloco de código?.

Exercício 4: Explique a diferença entre os tipos de dados int, float e str em Python.

Exercício 5: O que faz o operador ** em Python?

Exercício 6: Qual a diferença entre os operadores de comparação == e !=?

Exercício 7: Como transformar uma entrada de texto do usuário em um número inteiro?

Exercício 8: Qual é a diferença entre print() e input()?

Exercício 9: Escreva um programa que peça o nome do utilizador e o cumprimente.

Solução:

```
nome = input("Qual é o seu nome? ")
print("Olá,", nome, "! Bem-vindo ao Python!")
```

Exercício 10: Escreva um programa que peça dois números ao usuário e exiba a soma.

Solução:

```
numero1 = float(input("Digite o primeiro número: "))
numero2 = float(input("Digite o segundo número: "))
soma = numero1 + numero2
print("A soma é:", soma)
```

Exercício 11: Crie um programa que converta uma temperatura de Celsius para Fahrenheit. A fórmula de conversão é: $F = C * 9/5 + 32$

```
celsius = float(input("Digite a temperatura em Celsius: "))
fahrenheit = celsius * 9/5 + 32
print("A temperatura em Fahrenheit é:", fahrenheit)
```

Solução:

Exercício 12: Escreva um programa que calcule o Índice de Massa Corporal (IMC) de uma pessoa. A fórmula é: $IMC = \text{peso} / (\text{altura} ^\star 2)$

Solução:

```
peso = float(input("Digite o seu peso (kg): "))
altura = float(input("Digite a sua altura (m): "))
imc = peso / (altura ** 2)
print("Seu IMC é:", imc)
```

Exercício 13: Escreva um programa que peça o nome e o sobrenome do utilizador e o cumprimente.

Exercício 14: Escreva um programa que peça três números ao utilizador e exiba a soma.

Exercício 15: Crie um programa que receba dois números e exiba:

A soma, diferença, produto, e o quociente. Exemplo de entrada e saída

```
# Entrada
10, 5

# Saída
Soma: 15
Diferença: 5
Produto: 50
Divisão: 2.0
```

Exercício 16: Escreva um programa que pergunte a largura e altura de um retângulo e calcule sua área.

Exercício 17: Escreva um programa que peça dois números ao usuário e uma operação matemática (+, -, *, /). O programa deve realizar a operação e exibir o resultado.

Exercício 18: Escreva um programa que recebe dois valores inteiros e armazene-os cada um em uma variável, efectue a troca de valores entre as duas variáveis. No final ao imprimir o a primeira variável deve apresentar o valor que estava na segunda e vice-versa.

2 - Controle de Fluxo

O que é controle de fluxo?

O **controle de fluxo** é mecanismo permite que o fluxo do program seja desviado ou que um bloco de código seja repetido conforme necessário. Ele é usado para direcionar o comportamento do programa, determinando **o que executar** e **quando executar**. Em Python, isso é feito com:

- 1 - **Estruturas Condicionais:** `if, elif, else`.
- 2 - **Estruturas de Repetição:** `for, while`.
- 3 - **Controle Adicional em Loops:** `break, continue`.

2.1 - Estruturas Condicionais

Permitem que uma instrução seja executada mediante uma condição. A palavra chave `if` é usada para adicionar condições ao código.

Estrutura condicional simples:

```
if <condição>:  
    <instrução>
```

Exemplo:

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

Estrutura condicional composta:

```
if <condição>:  
    <instrução>  
else:  
    <instrução>
```

Exemplo:

```
idade = 15  
if idade >= 18:  
    print("Você é maior de idade.")  
else:  
    print("Você é menor de idade.")
```

Estrutura condicional de múltiplas escolhas:

```
if <condição>:  
    <instrução>  
elif <condição>:  
    <instrução>  
else:  
    <instrução>
```

Exemplo:

```
nota = float(input("Digite sua nota: "))  
if nota >= 7:  
    print("Aprovado")  
elif nota >= 5:  
    print("Recuperação")  
else:  
    print("Reprovado")
```

2.2 - Estruturas de repetição

Permitem executar uma instrução ou bloco de instrução de forma repetida. No python as estruturas de repetição são: **for, while**

for utilizado para percorrer sequências.

```
for <elemento_na_sequência> in <sequência>:  
    print(<elemento_na_sequência>)
```

Exemplo:

```
for i in range(1, 5):  
    print(i)
```

while utilizado para repetir uma instrução ou bloco de instrução enquanto a condição for verdadeira.

```
while <condição>:  
    <instrução>
```

Exemplo:

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

2.3 - Controle Adicional em Loops

Existem duas instruções que servem para controlar o fluxo de execução em loops. Elas são: `break` e `continue`.

O `break` serve para interromper o fluxo do loop imediatamente.

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

O `continue` serve para pular para a próxima iteração.

```
for i in range(1, 6):
    if i == 3:
        continue
    print(i)
# Saída: 1 2 4 5
```

2.4 - Compreensão sobre loops aninhados

O que são loops aninhados?

Loops aninhados (ou laços aninhados) são estruturas de repetição dentro de outras estruturas de repetição. Ou seja, um `for` ou `while` é colocado dentro de outro `for` ou `while`.

Quando usar loops aninhados?

Utilizamos loops aninhados quando queremos percorrer estruturas compostas, como:

- Matrizes (listas de listas);
- Tabelas de valores;
- Combinações de elementos;
- Impressões de padrões (como triângulos com asteriscos);
- Comparações entre todos os elementos de duas listas.

Como o fluxo de execução se comporta?

O loop externo executa uma vez, e para **cada execução** dele, o loop interno executa **completo**.

Exemplo simples:

```
for i in range(2):          # Executa 2 vezes
    for j in range(3):      # Executa 3 vezes para cada i
        print(f"i={i}, j={j}")
```

Saída:

```
i=0, j=0
i=0, j=1
i=0, j=2
i=1, j=0
i=1, j=1
i=1, j=2
```

Vamos entender o fluxo passo a passo:

1. $i=0 \rightarrow$ entra no segundo for

- $j=0$, imprime
- $j=1$, imprime
- $j=2$, imprime

2. $i=1 \rightarrow$ entra novamente no segundo for

- $j=0$, imprime
- $j=1$, imprime
- $j=2$, imprime

2.5 - Introdução às funções embutidas.

O que são funções embutidas (built-in functions)?

São funções que já vêm prontas no Python, sem necessidade de importação. `range()`, `len()`, `type()`, `abs()`, `round()`, `sorted()`, `int()`, `float()`, e `str()` são exemplos de funções embutidas. É importante saber que não são as únicas, confira a lista completa [em](#).

Casos de uso:

`range()` - gera uma sequência de números

```
for i in range(3):
    print(i)
# Saída: 0, 1, 2
# Usado com frequência em loops.
```

`len()` - retorna o comprimento (tamanho) de uma sequência

```
texto = "python"
print(len(texto)) # Saída: 6
```

`sorted()` - retorna uma nova lista com os elementos ordenados

```
numeros = [3, 1, 2]
print(sorted(numeros)) # Saída: [1, 2, 3]
print(numeros) # A lista original não é alterada
```

`float()` - retorna uma nova lista com os elementos ordenados

```
print(int("42")) # Saída: 42
```

2.6 - Exercícios sobre controlo de fluxo

1. Escreva um programa que peça um número inteiro ao utilizador e diga se ele é par ou ímpar.
2. Use um while para imprimir os números de 10 até 1 na tela.
3. Use um for para somar todos os números de 1 a 100 e imprimir o resultado.
4. Peça dois números ao utilizador e mostre qual deles é o maior (ou se são iguais).
5. Peça ao utilizador um número de 1 a 10 e imprima sua tabuada.
6. Faça um programa em que o utilizador tem 3 tentativas para digitar a senha correta. Use while e break.
7. Peça números ao utilizador e vá somando até que ele digite 0. Depois, mostre o total.
8. Peça dois números inteiros e imprima todos os números pares entre eles, inclusive os limites.
9. Crie um menu com as opções:

1: dizer "Olá"

2: dizer "Tchau"

0: sair

DICA: Use while para repetir até o utilizador escolher sair.

10. Peça uma nota de 0 a 10. Se for inválida, peça novamente até que esteja no intervalo.
11. Peça um número inteiro e diga se ele é primo ou não.
12. Peça ao utilizador um número e imprima um padrão assim (se o número for 4):

```
*  
***  
*****  
*****
```

13. Peça uma frase ao utilizador e mostre quantas vogais e quantas consoantes ela tem.
14. O programa escolhe um número entre 1 e 100. O utilizador deve adivinhar, e o programa dá dicas se está acima ou abaixo. Mostre o número de tentativas ao final.
15. O utilizador digita um valor e o programa deve dizer quantas notas de 100, 50, 20, 10, e 5 kwanzas são necessárias para formar esse valor.

3. Funções e Estruturas de dados básicas

3.1 - Definição e chamada de funções.

O que é um método ou função?

Uma **função** (ou método) é um bloco de código reutilizável que executa uma tarefa específica.

- Em Python, usamos a palavra-chave `def` para criar funções.
- O termo "método" costuma ser usado quando a função está dentro de uma classe.

Exemplo de criação de função:

```
def saudacao():
    print("Olá, bem-vindo!")
```

Aqui estamos definindo a função chamada `saudacao`. Ela **não recebe parâmetros** e apenas **exibe uma mensagem**.

Para chamar uma basta usar o nome da função seguido de parênteses. Se ela tiver parâmetros, você deve passá-los nos parênteses.

```
saudacao()
```

3.2 - Parâmetros e retorno de valores

O que é um parâmetro?

Parâmetros são **variáveis** que você define entre os parênteses da função. Elas servem para **passar valores** para dentro da função.

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

Aqui, `nome` é um parâmetro. A função depende dele para funcionar corretamente.

PARÂMETROS OBRIGATÓRIOS VS OPCIONAIS

Parâmetros obrigatórios devem ser informados ao chamar a função.

```
def mostrar_idade(idade):
    print(f"Você tem {idade} anos.")
```

Ao passo que parâmetros opcionais têm um valor padrão e podem ser omitidos ao chamar a função.

```
def boas_vindas(nome="Visitante"):
    print(f"Bem-vindo, {nome}!")
```

CHAMADAS POSSÍVEIS:

```
boas_vindas()           # usa o padrão
boas_vindas("Mariana") # substitui o padrão
```

O QUE É O RETORNO DE UMA FUNÇÃO?

É o **valor que a função entrega de volta** para quem a chamou, usando `return`.

```
def soma(a, b):
    return a + b
```

Exemplo de uso:

```
resultado = soma(3, 5)
print(resultado) # Saída: 8
```

OBS: Se uma função não tiver `return`, ela retorna `None` por padrão.

Dado o seguinte exemplo, responde:

```
def calcular_desconto(preco, percentual=10):
    desconto = preco * percentual / 100
    return preco - desconto
```

- qual o nome da função?
- quantos argumentos tem a função?
- qual deles é obrigatório?
- qual é opcional?
- A função tem retorno?

3.3 - Introdução às listas, sets e dicionários: criação, indexação e manipulação.

3.3.1 - Listas

Uma **lista** é uma coleção **ordenada** e **mutável** de elementos. Pode conter qualquer tipo de dado (números, strings, outras listas, etc.).

3.3.1.1 - Como criar uma lista?

Para criar listas usamos parênteses retos [], os elementos da lista devem ficar dentro deste parênteses separados por vírgula.

Exemplo:

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

3.3.1.2 - Como acessar (indexar) elementos?

Acessámos os elementos de listas indexando a lista. Os índices de uma lista começam em zero (0) e termina na quantidade de elementos menos um. No exemplo acima o último elemento é 2.

Exemplo:

```
print(frutas[0]) # maçã
print(frutas[2]) # laranja
```

3.3.1.3 - Como manipular uma lista?

Existem alguns métodos pré definidos para manipulação de listas como `append()` e `remove()`. Para mais detalhes ver [em](#).

Exemplo:

```
frutas.append("abacaxi") # adiciona no final
frutas.remove("banana") # remove o valor
frutas[1] = "uva"       # altera valor no índice 1
```

3.3.2 - Sets (Conjuntos)

Um **set** é uma coleção **não ordenada e sem elementos duplicados**. Muito útil para verificar existência e remover duplicatas.

3.3.2.1 - Como criar sets?

Para criar sets usamos chavetas {}, os elementos da lista devem ficar dentro destas chavetas separados por vírgula.

Exemplo:

```
numeros = {1, 2, 3, 3, 2}
print(numeros) # {1, 2, 3} – duplicatas removidas
```

3.3.2.2 - Como acessar elementos de um set?

Não é possível indexar sets diretamente (por serem desordenados), mas é possível iterar sobre ele.

```
for num in numeros:
    print(num)
```

3.3.2.3 - Como manipular uma lista?

Existem alguns métodos pré definidos para manipulação de sets como **add()** e **remove()**.

Exemplo:

```
numeros.add(4)
numeros.remove(2)
```

3.3.3 - Dicionários

Um **dicionário** é uma coleção de **pares chave:valor**. Muito útil para armazenar e acessar dados nomeados.

3.3.3.1 - Como criar sets?

Para criar dicionários também usamos chavetas como em sets, a grande diferença é que cada elemento de um dicionário é um par de **chave** e **valor** separados por dois pontos :

Exemplo:

```
pessoa = {  
    "nome": "João",  
    "idade": 30,  
    "cidade": "São Paulo"  
}
```

3.3.3.2 - Como acessar valores por chave?

Para acessar um valor do dicionário devemos indexar (assim como em listas) o dicionário especificando a sua chave.

```
print(pessoa["nome"]) # João
```

3.3.3.4. - Como manipular um dicionário?

```
pessoa["idade"] = 31           # altera valor  
pessoa["profissão"] = "Engenheiro" # adiciona novo par  
del pessoa["cidade"]          # remove chave
```