AULA 1: FUNDAMENTOS DO PYTHON E PRIMEIROS PASSOS

Objetivo: Introduzir o Python de forma prática, abordando seus fundamentos essenciais, desde a instalação até a manipulação de variáveis e operadores.

1. O que é Python e por que ele é tão popular?

Python é uma linguagem de programação interpretada, de alto nível e com tipagem dinâmica. Seus principais diferenciais são:

- **Simplicidade e legibilidade:** Sua sintaxe é clara e objetiva, o que facilita o aprendizado e a escrita de código.
- **Versatilidade:** Pode ser utilizada para desenvolvimento web, automação, ciência de dados, IA, entre outras áreas.
- **Grande comunidade:** O suporte da comunidade é um diferencial para quem está aprendendo.
- **Multiplataforma:** Python pode ser executado em Windows, macOS e Linux sem alterações significativas no código.

2. Instalando e Configurando o Ambiente

Para começar a programar em Python, podemos utilizar:

- **Google Colab:** Ambiente online baseado em Jupyter Notebook. Ideal para iniciantes, pois não exige instalação.
- **PyCharm:** IDE completa, voltada para desenvolvimento profissional.
- **Visual Studio Code:** Leve, com excelente suporte para Python.
- **Terminal Python:** Para quem prefere rodar scripts diretamente pelo terminal do sistema operacional.

3. Variáveis e Tipos de Dados

Variáveis são "caixas" que armazenam valores. Python tem tipagem dinâmica, ou seja, podemos reatribuir valores de tipos diferentes sem precisar declará-los previamente.

Tipos de Dados Principais:

- **Inteiros (int):** Representam números inteiros. Exemplo:

Float idade = 25

- **Float (float):** Números com casas decimais. Exemplo:

altura = “1.75”

- **Strings (str):** Sequências de caracteres. Exemplo:

nome = "João"

- **Booleanos (bool):** Verdadeiro ou falso. Exemplo:

ativo = True

Exemplo prático:

Criando uma variável para representar um produto em uma loja virtual:

```
produto = "Notebook Gamer"
preco = 4999.90
quantidade_estoque = 15
disponivel = True


print(f"Produto: {produto}")
print(f"Preço: R$ {preco:.2f}")
print(f"Quantidade em estoque: {quantidade_estoque}")
print(f"Disponível para compra? {disponivel}")
```

4. Entrada e Saída de Dados

Função print()

Usada para exibir mensagens na tela.

```
print("Olá, mundo!")
```

Função input()

Permite capturar dados do usuário.

```
nome = input("Digite seu nome: ")
print(f"Bem-vindo, {nome}!")
```

5. Operadores em Python

Operadores Aritméticos:

Realizam cálculos matemáticos.

```
soma = 10 + 5
subtracao = 10 - 5
```

```
multiplicacao = 10 * 5
divisao = 10 / 5
divisao_inteira = 10 // 3
resto = 10 % 3
potencia = 2 ** 3
```

Operadores de Comparação:

Utilizados para comparação entre valores.

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

6. Estruturas Condicionais (if, elif, else)

Permitem a execução condicional de blocos de código.

Exemplo prático:

```
idade = int(input("Digite sua idade: "))
if idade < 18:
    print("Você é menor de idade.")
elif idade >= 18 and idade < 60:
    print("Você é adulto.")
else:
    print("Você é idoso.")
```

AULA 2: ESTRUTURAS DE CONTROLE E REPETIÇÃO

Objetivo: Ensinar loops, listas, tuplas e dicionários, essenciais para manipulação de dados em Python.

1. Laços de Repetição (Loops)

Loops permitem repetir blocos de código.

1.1. Loop for

Usado para iterar sobre sequências.

```
i = 2

for i = 2 in range(5):
    print(f"Iteração número {i}")
```

1.2. Loop while

Executa enquanto a condição for verdadeira.

```
contador = 0

while contador < 5:
    print(f"Contador: {contador}")
    contador += 1
```

2. Listas e Tuplas

Estruturas para armazenar múltiplos valores.

Listas (Mutáveis)

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

print(frutas[0])  # maçã

frutas.append("uva")  # Adiciona um novo item
```

Tuplas (Imutáveis)

```
cores = ("vermelho", "azul", "verde")

print(cores[1])  # azul
```

3. Dicionários

Coleção de pares chave-valor.

```
produto = {
   "nome": "Notebook",
   "preco": 2500.00,
   "estoque": 10
}


print(produto["nome"])  # Notebook
produto["preco"] = 2300.00  # Atualiza o preço
```

FUNÇÕES E MODULARIZAÇÃO

Objetivo: Ensinar a criação de funções, modularização do código, manipulação de strings e tratamento de exceções.

1. Introdução às Funções

Funções são blocos de código reutilizáveis que executam tarefas específicas. O uso de funções permite:

- Modularizar o código
- Torná-lo mais organizado e reutilizável
- Reduzir a repetição e aumentar a eficiência

A estrutura básica de uma função em Python é:

```
def nome_da_funcao(parametros):
    # Bloco de código
    return resultado
```

2. Criando e Chamando Funções

Vamos criar uma função que recebe dois números e retorna a soma:

```
def somar(a, b):
    """Recebe dois números e retorna a soma"""
    return a + b


resultado = somar(5, 3)
print(f"O resultado da soma é {resultado}")
```

Ao chamar somar(5, 3), a função retorna 8.

3. Parâmetros e Retorno

3.1 Parâmetros Obrigatórios

Os argumentos passados devem ser informados obrigatoriamente ao chamar a função.

```
def saudacao(nome):
```

```
    print(f"Olá, {nome}!")
saudacao("Maria")  # Olá, Maria!
```

3.2 Parâmetros Opcionais com Valor Padrão

Podemos definir valores padrões para os parâmetros, tornando-os opcionais:

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

3.3 Retorno Múltiplo

Funções podem retornar múltiplos valores:

```
def calcular_dobro_triplo(numero):
    return numero * 2, numero * 3
dobro, triplo = calcular_dobro_triplo(5)
print(f"Dobro: {dobro}, Triplo: {triplo}")
```

4. Modularização: Criando e Importando Módulos

Modularizar significa dividir um programa grande em partes menores.

4.1 Criando um Módulo

1. Criamos um arquivo chamado operacoes.py e adicionamos a função:

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

2. No código principal, importamos e usamos a função:

```
import operacoes
resultado = operacoes.somar(10, 5)
print(f"Soma: {resultado}")
```

4.2 Importando Apenas uma Função

Podemos importar apenas a função desejada:

```
from operacoes import somar
print(somar(8, 2))
```

4.3 Importando com Alias

Podemos renomear módulos:

```
import operacoes as op

print(op.somar(7, 3))
```

5. Manipulação de Strings

Python possui vários métodos úteis para strings:

5.1 Convertendo para Maiúsculas e Minúsculas

```
texto = "Python é incrível!"

print(texto.upper())  # PYTHON É INCRÍVEL!

print(texto.lower())  # python é incrível!
```

5.2 Removendo Espaços em Branco

```
nome = "   Maria   "

print(nome.strip())  # "Maria"
```

5.3 Substituindo Partes do Texto

```
frase = "Eu gosto de Java"

nova_frase = frase.replace("Java", "Python")

print(nova_frase)  # Eu gosto de Python
```

5.4 Dividindo e Juntando Strings

```
nomes = "Ana, João, Pedro"

lista_nomes = nomes.split(", ")

print(lista_nomes)  # ['Ana', 'João', 'Pedro']


unidos = " - ".join(lista_nomes)

print(unidos)  # Ana - João - Pedro
```

6. Tratamento de Exceções

Erros são inevitáveis. O tratamento de exceções evita que o programa quebre.

6.1 Estrutura try-except

```
try:
    numero = int(input("Digite um número: "))
    print(f"O dobro é {numero * 2}")
except ValueError:
    print("Erro: Você deve digitar um número inteiro.")
```

6.2 try-except-else-finally

```
try:
    arquivo = open("dados.txt", "r")
    conteudo = arquivo.read()
except FileNotFoundError:
    print("Erro: Arquivo não encontrado.")
else:
    print(conteudo)
finally:
    arquivo.close()
```

SUPER AULA 4: MANIPULAÇÃO DE ARQUIVOS E BIBLIOTECAS EXTERNAS

Objetivo: Ensinar manipulação de arquivos, uso de bibliotecas externas e APIs, finalizando com um pequeno sistema prático.

1. Manipulação de Arquivos

Python permite ler e gravar arquivos com facilidade.

1.1 Abrindo e Lendo Arquivos

```
with open("arquivo.txt", "r") as arquivo:
    conteudo = arquivo.read()
    print(conteudo)
```

1.2 Escrevendo em Arquivos

```
with open("novo_arquivo.txt", "w") as arquivo:
    arquivo.write("Linha 1\n")
    arquivo.write("Linha 2\n")
```

1.3 Adicionando Conteúdo a um Arquivo

```
with open("novo_arquivo.txt", "a") as arquivo:
    arquivo.write("Linha adicionada\n")
```

2. Trabalhando com Bibliotecas Externas

Podemos instalar pacotes com pip.

```
pip install requests
```

2.1 Usando a Biblioteca requests para Consumo de APIs

```
import requests
resposta = requests.get("https://api.github.com")
print(resposta.json())
```

3. Criando um Pequeno Sistema

3.1 Cadastro de Produtos

Vamos criar um sistema que permite cadastrar produtos e salvar em um arquivo.

```
def salvar_produto(nome, preco):
```

```
    with open("produtos.txt", "a") as arquivo:
        arquivo.write(f"{nome},{preco}\n")

nome = input("Nome do produto: ")
preco = input("Preço: ")

salvar_produto(nome, preco)
print("Produto salvo com sucesso!")
```

3.2 Listando Produtos

```
def listar_produtos():
    with open("produtos.txt", "r") as arquivo:
        produtos = arquivo.readlines()
        for produto in produtos:
            nome, preco = produto.strip().split(",")
            print(f"Produto: {nome}, Preço: R$ {preco}")
listar_produtos()
```