
Autoria: Augusto Bemfica Mombach

# Apostila de Python

## Capítulo 1: Introdução ao Python

## 1.1 Sobre Python

O que é Python?

Python é uma linguagem de programação de alto nível, interpretada e com uma sintaxe que privilegia a legibilidade do código. Desenvolvida por Guido van Rossum e lançada pela primeira vez em 1991, Python rapidamente se tornou popular devido à sua simplicidade e versatilidade.

Características Principais:

- Sintaxe Simples e Legível: Python é conhecida pela sua sintaxe clara e fácil de entender, o que facilita a aprendizagem e a manutenção do código.
- Tipagem Dinâmica: Não é necessário declarar o tipo de uma variável, o Python faz isso automaticamente durante a execução do código.
- Portabilidade: Python é uma linguagem multiplataforma, podendo ser executada em Windows, macOS, Linux, entre outros.
- Bibliotecas Extensas: Uma vasta gama de bibliotecas e frameworks está disponível, facilitando desde o desenvolvimento web até a análise de dados e machine learning.

Aplicações de Python:

- Desenvolvimento Web (com frameworks como Django e Flask)
- Análise de Dados e Ciência de Dados (usando bibliotecas como Pandas e NumPy)
- Inteligência Artificial e Machine Learning (com ferramentas como TensorFlow e Scikit-learn)
- Automação de Tarefas e Scripting

## 1.2 Comunidade e Recursos


Comunidade Ativa: A comunidade Python é uma das mais ativas e acolhedoras, oferecendo suporte através de fóruns, conferências e meetups.

Recursos para Aprendizado:

- Documentação Oficial: python.org
- Tutoriais Online: Sites como Codecademy, Coursera e Udemy oferecem cursos de Python.
- Livros: "Automatize Tarefas Maçantes com Python", "Python Fluente" e "Aprendendo Python" são excelentes fontes de estudo.

---

# Capítulo 2: Configuração do Ambiente de Desenvolvimento

## 2.1 Instalação do Python

Windows:

Acesse python.org
Baixe a versão mais recente do Python para Windows.
Execute o instalador. Importante: Marque a opção "Add Python to PATH" durante a instalação.
Após a instalação, abra o prompt de comando e digite `python` para verificar a instalação.

macOS:

Python geralmente vem pré-instalado no macOS. Para atualizar ou instalar, você pode usar o Homebrew com o comando `brew install python`.
Verifique a instalação abrindo o Terminal e digitando `python3`.

Linux:

Python geralmente já está instalado na maioria das distribuições Linux. Para verificar, abra o terminal e digite `python3`.
Se necessário, instale ou atualize o Python usando o gerenciador de pacotes da sua distribuição (como `apt` ou `yum`).

## 2.2 Configuração de um Editor de Código ou IDE


IDEs e Editores Recomendados:

- Visual Studio Code: Leve e altamente personalizável. Suporta Python através de extensões.
- PyCharm: Uma IDE específica para Python, oferece uma versão gratuita (Community) e uma versão paga com mais recursos.
- Jupyter Notebook: Ótimo para aprendizado e análise de dados, permite escrever e executar o código em blocos.

Instalação do Visual Studio Code:

Baixe o Visual Studio Code de code.visualstudio.com
Instale a extensão Python pela loja de extensões do VSCode.
Configure o interpretador Python selecionando a versão instalada na sua máquina.

## 2.3 Executando o Primeiro Script Python

Criando e Executando um Script:

Abra o editor de código.
Crie um novo arquivo com a extensão `.py`, por exemplo, `hello.py`.
Escreva o seguinte código: `print("Olá, mundo!")`
Execute o script no terminal ou no próprio editor (se suportado) usando `python hello.py`.

# Capítulo 3: Fundamentos de Python

## 3.1 Primeiro Programa em Python

A primeira etapa no aprendizado de qualquer linguagem de programação é escrever o tradicional "Hello, World!". Em Python, isso é feito de maneira bastante simples.

Exemplo de Código:

```
print("Olá, mundo!")
```

Explicação:

- `print()` é uma função embutida em Python que exibe o conteúdo dentro dos parênteses.


- `"Olá, mundo!"` é uma string, um tipo de dado em Python usado para representar texto.

## 3.2 Variáveis e Tipos de Dados

As variáveis são usadas para armazenar informações que podem ser referenciadas e manipuladas em programas. Python é uma linguagem de tipagem dinâmica, o que significa que não é necessário declarar o tipo de variável.

Tipos de Dados Básicos:

Inteiros e Flutuantes:

- Números inteiros (sem ponto decimal) e flutuantes (com ponto decimal). Exemplo:

```
numero_inteiro = 10
numero_flutuante = 3.14
```

Strings:

- Sequências de caracteres usadas para armazenar texto. Exemplo:

```
texto = "Aprendendo Python"
```

Booleanos:

- Representa verdadeiro ou falso, útil para condições e decisões. Exemplo:

```
verdadeiro = True
falso = False
```

Operações com Variáveis:

- Python suporta operações comuns como adição (`+`), subtração (`-`), multiplicação (`*`) e divisão (`/`). Exemplo de Operações:

```
soma = numero_inteiro + 5

concatenação = texto + " é divertido!"
```

---

# Capítulo 4: Estruturas de Controle em Python

Estruturas de controle permitem que você direcione o fluxo de execução do seu programa com base em condições e repetições.

## 4.1 Condicionais

Usadas para executar ações diferentes com base em diferentes condições.

Exemplo de Código:

```
idade = 18
if idade < 18:
 print("Menor de idade")
elif idade == 18:
 print("Tem exatamente 18 anos")
else:
 print("Maior de idade")
```

Explicação:

- `if`, `elif` e `else` permitem a execução condicional de blocos de código.
- A identação é crucial em Python para definir blocos de código.

## 4.2 Loops

Loops são usados para repetir um bloco de código várias vezes.

While Loop:

- Repete um bloco de código enquanto uma condição for verdadeira. Exemplo:

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

For Loop:

- Itera sobre uma sequência (como uma lista, tupla ou string). Exemplo:

```
for i in range(5):
 print("Valor de i =", i)
```

Controle de Loop:

- `break` pode ser usado para sair de um loop.
- `continue` pula para a próxima iteração do loop.

# Capítulo 5: Entrada e Saída de Dados

A interação com o usuário é um aspecto fundamental em muitos programas. Em Python, você pode facilmente receber a entrada do usuário e exibir saídas usando as funções embutidas `input()` e `print()`.

## 5.1 Saída de Dados com `print()`

A função `print()` é usada para enviar dados para a saída padrão (geralmente a tela).

Exemplos de Uso:

- Imprimindo texto simples:

```
print("Olá, mundo!")
```

- Imprimindo variáveis:

```
mensagem = "Aprendendo Python"
print(mensagem)
```

Formatação de Strings:

- Concatenação:

```
nome = "Ana"
print("Olá " + nome)
```

- Formatação com `f-strings` (Python 3.6+):

```
idade = 30
```

```
print(f"Tenho {idade} anos")
```

## 5.2 Entrada de Dados com `input()`

A função `input()` pausa a execução do programa e espera que o usuário digite algo. Após a entrada, pressionar Enter continuará a execução do programa.

Exemplos de Uso:

- Recebendo texto do usuário:

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

- Recebendo números (lembrar de converter a entrada para o tipo numérico):

```
idade = int(input("Digite sua idade: "))
print(f"Você tem {idade} anos.")
```

## 5.3 Manipulando Entradas e Saídas

Validação de Entrada:

- É importante validar as entradas do usuário para evitar erros. Por exemplo, ao esperar um número, certifique-se de que o usuário digitou de fato um número.

```
try:
 numero = int(input("Digite um número: "))
 print(f"O número é {numero}")
except ValueError:
 print("Não é um número válido!")
```

Uso de Loops com `input()`:

- Você pode usar `input()` dentro de um loop para continuar recebendo dados até que uma condição seja atendida.

```
resposta = ""
while resposta.lower() != "sair":
 resposta = input("Digite algo (ou 'sair' para terminar): ")
```

```
print(f"Você digitou: {resposta}")
```

# Lista de Exercícios - Python Básico

## Exercício 1: "Hello, World!"

Escreva um programa em Python que imprima "Hello, World!" na tela.

## Exercício 2: Manipulação de Variáveis

Crie variáveis para armazenar seu nome, idade e cidade onde mora. Em seguida, imprima essas informações em uma frase completa.

## Exercício 3: Operações Matemáticas

Peça ao usuário para inserir dois números e imprima a soma, subtração, multiplicação e divisão desses números.

## Exercício 4: Maior ou Menor

Escreva um programa que compare dois números inseridos pelo usuário e imprima qual é o maior ou se são iguais.

## Exercício 5: Verificador de Paridade

Peça ao usuário para inserir um número e verifique se é par ou ímpar.

## Exercício 6: Cálculo da Média

Escreva um programa que calcule a média de 5 números fornecidos pelo usuário.

## Exercício 7: Loop de Contagem

Use um loop para imprimir todos os números de 1 a 10.

## Exercício 8: Soma dos Números

Utilize um loop para calcular a soma de todos os números de 1 a 100.


## Exercício 9: Tabuada

Peça ao usuário para inserir um número e imprima a tabuada desse número (de 1 a 10).

## Exercício 10: Adivinhação

Faça um programa que escolhe um número aleatório entre 1 e 10 e peça ao usuário para tentar adivinhá-lo. Dê dicas se o palpite for maior ou menor que o número escolhido.

# Soluções dos Exercícios - Python Básico

## Exercício 1: "Hello, World!"

```
print("Hello, World!")
```

## Exercício 2: Manipulação de Variáveis

```
nome = "João"
idade = 30
cidade = "São Paulo"
print(f"Meu nome é {nome}, tenho {idade} anos e moro em {cidade}.")
```

## Exercício 3: Operações Matemáticas

```
num1 = float(input("Digite o primeiro número: "))
num2 = float(input("Digite o segundo número: "))

print(f"Soma: {num1 + num2}")
print(f"Subtração: {num1 - num2}")
print(f"Multiplicação: {num1 * num2}")
print(f"Divisão: {num1 / num2}")
```

## Exercício 4: Maior ou Menor

```
numero1 = float(input("Digite o primeiro número: "))
```

```
numero2 = float(input("Digite o segundo número: "))

if numero1 > numero2:
 print("O primeiro número é maior.")
elif numero2 > numero1:
 print("O segundo número é maior.")
else:
 print("Os números são iguais.")
```

## Exercício 5: Verificador de Paridade

```
numero = int(input("Digite um número: "))

if numero % 2 == 0:
 print("O número é par.")
else:
 print("O número é ímpar.")
```

## Exercício 6: Cálculo da Média

```
total = 0
for i in range(5):
 numero = float(input(f"Digite o número {i + 1}: "))
 total += numero

media = total / 5
print(f"A média é {media}")
```

## Exercício 7: Loop de Contagem

```
for i in range(1, 11):
 print(i)
```

## Exercício 8: Soma dos Números

```
soma = sum(range(1, 101))
print(f"A soma é {soma}")
```

## Exercício 9: Tabuada

```
numero = int(input("Digite um número para ver sua tabuada: "))
for i in range(1, 11):
 print(f"{numero} x {i} = {numero * i}")
```

## Exercício 10: Adivinhação

```
import random

numero_secreto = random.randint(1, 10)
tentativa = 0

while tentativa != numero_secreto:
 tentativa = int(input("Adivinhe um número entre 1 e 10: "))
 if tentativa > numero_secreto:
 print("Menor")
 elif tentativa < numero_secreto:
 print("Maior")

print("Parabéns! Você acertou.")
```

# Capítulo 6: Funções em Python

As funções são blocos de código que podem ser reutilizados e têm como objetivo realizar uma tarefa específica. Em Python, funções são definidas usando a palavra-chave `def`.

## 6.1 Definindo e Chamando Funções

Exemplo de Função Simples:

```
def cumprimentar():
 print("Olá, bem-vindo ao curso de Python!")
```

## 6.2 Parâmetros e Argumentos


Parâmetros são variáveis listadas na definição da função. Argumentos são valores passados para a função.

Exemplo com Parâmetros:

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

cumprimentar("Maria") # Passando "Maria" como argumento
```

## 6.3 Retorno de Valores

Uma função pode retornar um valor usando a palavra-chave `return`.

Exemplo de Função com Retorno:

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

resultado = somar(5, 3)
print(f"Resultado: {resultado}")
```

## 6.4 Funções com Vários Parâmetros

Funções podem ter vários parâmetros, permitindo maior flexibilidade.

Exemplo de Função com Múltiplos Parâmetros:

```
def descrever_pessoa(nome, idade):
 print(f"Nome: {nome}, Idade: {idade}")

descrever_pessoa("Carlos", 40)
```

---

# Capítulo 7: Estruturas de Dados em Python

Python oferece várias estruturas de dados embutidas, como listas, dicionários, tuplas e conjuntos, que são usadas para armazenar coleções de dados.

## 7.1 Listas

Listas são usadas para armazenar múltiplos itens em uma única variável. São mutáveis, o que significa que podem ser alteradas após sua criação.

Exemplo de Lista:

```
frutas = ["maçã", "banana", "cereja"]
print(frutas)
```

## 7.2 Dicionários

Dicionários armazenam pares de chave-valor e são otimizados para recuperar valores quando a chave é conhecida.

Exemplo de Dicionário:

```
pessoa = {"nome": "Ana", "idade": 25}
print(pessoa["nome"]) # Acessando o valor associado à chave "nome"
```

## 7.3 Tuplas

Tuplas são semelhantes às listas, mas são imutáveis. Uma vez que uma tupla é criada, você não pode alterar seus valores.

Exemplo de Tupla:

```
dimensoes = (20, 50)
print(dimensoes)
```

## 7.4 Conjuntos

Conjuntos são coleções desordenadas e não indexadas. Eles são úteis para testar a presença de um elemento e eliminar duplicatas.

Exemplo de Conjunto:

```
numeros = {1, 2, 3, 4, 5}
print(numeros)
```

# Lista de Exercícios - Funções e Estruturas de Dados em Python

## Exercício 1: Função de Saudação

Escreva uma função chamada `saudacao` que receba um nome como parâmetro e imprima "Olá [nome], seja bem-vindo!".

## Exercício 2: Calculadora Simples

Crie uma função `calculadora` que aceite dois números e uma operação (soma, subtração, multiplicação, divisão) e retorne o resultado da operação.

## Exercício 3: Contador

Desenvolva uma função que receba um número como parâmetro e imprima todos os números de 1 até esse número.

## Exercício 4: Verificador de Ano Bissexto

Escreva uma função que verifique se um ano (passado como parâmetro) é bissexto ou não.

## Exercício 5: Máximo e Mínimo

Faça uma função que receba uma lista de números e retorne o maior e o menor número da lista.

## Exercício 6: Manipulação de Listas

Dada uma lista de números, escreva um programa que imprima:

- Todos os números (usando um loop)
- O maior número
- O menor número

## Exercício 7: Frequência dos Elementos

Escreva uma função que conte a frequência de cada elemento em uma lista e imprima um dicionário com essas contagens.

## Exercício 8: Tuplas para String

Crie uma função que converta uma tupla de caracteres em uma string.

## Exercício 9: União de Conjuntos

Dados dois conjuntos, escreva um programa que retorne a união desses conjuntos.

## Exercício 10: Chaves e Valores

Dado um dicionário, escreva um programa que imprima todas as chaves e todos os valores separadamente.

# Soluções dos Exercícios - Funções e Estruturas de Dados em Python

## Exercício 1: Função de Saudação

```
def saudacao(nome):

 print(f"Olá {nome}, seja bem-vindo!")

saudacao("João")
```

## Exercício 2: Calculadora Simples

```
def calculadora(num1, num2, operacao):
 if operacao == 'soma':
 return num1 + num2
 elif operacao == 'subtração':
 return num1 - num2
 elif operacao == 'multiplicação':
 return num1 * num2
 elif operacao == 'divisão':
 return num1 / num2

resultado = calculadora(10, 5, 'soma')
```

```
print(resultado)
```

## Exercício 3: Contador

```
def contador(limite):
 for i in range(1, limite + 1):
 print(i)

contador(5)
```

## Exercício 4: Verificador de Ano Bissexto

```
def eh_bissexto(ano):
 if (ano % 4 == 0 and ano % 100 != 0) or (ano % 400 == 0):
 return True
 else:
 return False

print(eh_bissexto(2020))
```

## Exercício 5: Máximo e Mínimo

```
def maximo_minimo(lista):
 return max(lista), min(lista)

resultado = maximo_minimo([1, 2, 3, 4, 5])
print(f"Máximo: {resultado[0]}, Mínimo: {resultado[1]}")
```

## Exercício 6: Manipulação de Listas

```
numeros = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5]

for numero in numeros:
 print(numero)

print("Maior número:", max(numeros))
print("Menor número:", min(numeros))
```

## Exercício 7: Frequência dos Elementos

```
def frequencia(lista):
 freq = {}
 for item in lista:
 freq[item] = freq.get(item, 0) + 1
 return freq

print(frequencia(['a', 'b', 'a', 'c', 'b', 'a', 'c']))
```

## Exercício 8: Tuplas para String

```
def tupla_para_string(tupla):
 return ''.join(tupla)

resultado = tupla_para_string(('P', 'y', 't', 'h', 'o', 'n'))
print(resultado)
```

## Exercício 9: União de Conjuntos

```
conjunto1 = {1, 2, 3}
conjunto2 = {3, 4, 5}

uniao = conjunto1.union(conjunto2)
print(uniao)
```

## Exercício 10: Chaves e Valores

```
dicionario = {'nome': 'Ana', 'idade': 25, 'cidade': 'São Paulo'}

print("Chaves:", dicionario.keys())
print("Valores:", dicionario.values())
```

# Capítulo 8: Introdução à Programação Orientada a Objetos (POO)

# Seção 1: Conceitos Básicos de POO

Definição de Objetos e Classes

Na programação orientada a objetos, os conceitos fundamentais são objetos e classes. Um objeto é uma instância de uma classe, que é um modelo que define atributos e comportamentos.

Exemplo:

```
# Definindo uma classe
class Carro:
 def __init__(self, marca, modelo):
 self.marca = marca
 self.modelo = modelo

 def descricao(self):
 return f"{self.marca} {self.modelo}"

# Criando objetos
carro1 = Carro("Toyota", "Corolla")
carro2 = Carro("Tesla", "Model S")

# Acessando atributos e métodos
print(carro1.descricao()) # Saída: Toyota Corolla
print(carro2.descricao()) # Saída: Tesla Model S
```

Atributos e Métodos

Atributos são variáveis pertencentes a um objeto, enquanto métodos são funções associadas a um objeto.

Exemplo:

```
class Pessoa:
 def __init__(self, nome, idade):
 self.nome = nome
 self.idade = idade

 def saudacao(self):
 return f"Olá, meu nome é {self.nome} e tenho {self.idade} anos."

# Criando um objeto e acessando atributos e métodos
```


```
pessoa1 = Pessoa("João", 30)
print(pessoa1.nome) # Saída: João
print(pessoa1.saudacao()) # Saída: Olá, meu nome é João e tenho 30 anos.
```

## Seção 2: Herança e Polimorfismo

Herança

Herança é um conceito em que uma classe pode herdar atributos e métodos de outra classe, permitindo a reutilização de código.

Exemplo:

```
class Animal:
 def __init__(self, nome):
 self.nome = nome

 def som(self):
 pass # Método a ser implementado nas classes filhas

class Cachorro(Animal):
 def som(self):
 return "Au Au!"

class Gato(Animal):
 def som(self):
 return "Miau!"

# Criando objetos e acessando métodos
dog = Cachorro("Rex")
cat = Gato("Bola")
print(dog.som()) # Saída: Au Au!
print(cat.som()) # Saída: Miau!
```

Polimorfismo

Polimorfismo permite que objetos de diferentes classes sejam tratados de maneira uniforme, usando métodos com o mesmo nome, mas comportamentos diferentes.

Exemplo:

```
def fazer_som(animal):
```

```
 return animal.som()

# Utilizando a função com diferentes objetos
print(fazer_som(dog)) # Saída: Au Au!
print(fazer_som(cat)) # Saída: Miau!
```

# Capítulo 9: Avançando em Programação Orientada a Objetos em Python

## Seção 1: Encapsulamento e Métodos Especiais

Encapsulamento

O encapsulamento em Python é realizado usando convenções, como o uso de métodos e atributos com underscore (convenção para indicar que são "privados").

Exemplo:

```
class ContaBancaria:
 def __init__(self, saldo):
 self._saldo = saldo

 def deposito(self, valor):
 self._saldo += valor

 def _saque(self, valor):
 self._saldo -= valor

 def get_saldo(self):
 return self._saldo

# Utilizando métodos e atributos encapsulados
conta = ContaBancaria(100)
conta.deposito(50)
conta._saque(30) # Não é recomendado acessar diretamente métodos privados
print(conta.get_saldo()) # Saída: 120
```

Métodos Especiais (Mágicos)

Em Python, os métodos especiais ou métodos mágicos são utilizados para sobrecarregar operadores ou comportamentos específicos de objetos.

Exemplo:

```
class Ponto:
 def __init__(self, x, y):
 self.x = x
 self.y = y

 def __str__(self):
 return f"Ponto ({self.x}, {self.y})"

 def __add__(self, outro_ponto):
 return Ponto(self.x + outro_ponto.x, self.y + outro_ponto.y)

# Utilizando métodos especiais
ponto1 = Ponto(1, 2)
ponto2 = Ponto(3, 4)
print(ponto1) # Saída: Ponto (1, 2)
resultado = ponto1 + ponto2
print(resultado) # Saída: Ponto (4, 6)
```

## Seção 2: Composição e Agregação

Composição

Composição é um conceito onde um objeto é composto por outros objetos.

Exemplo:

```
class Motor:
 def __init__(self, tipo):
 self.tipo = tipo

class Carro:
 def __init__(self, marca, motor):
 self.marca = marca
 self.motor = motor

 def descricao(self):
 return f"{self.marca} com motor {self.motor.tipo}"
```

```
# Utilizando composição
motor_do_carro = Motor("V8")
carro = Carro("Ferrari", motor_do_carro)
print(carro.descricao()) # Saída: Ferrari com motor V8
```

Agregação

Agregação é uma forma de relação entre objetos onde um objeto é parte de outro, mas pode existir independentemente.

Exemplo:

```
class Roda:
 def __init__(self, tamanho):
 self.tamanho = tamanho

class Carro:
 def __init__(self, marca, roda):
 self.marca = marca
 self.roda = roda

 def descricao(self):
 return f"{self.marca} com rodas de tamanho {self.roda.tamanho}"

# Utilizando agregação
roda_do_carro = Roda(18)
carro = Carro("BMW", roda_do_carro)
print(carro.descricao()) # Saída: BMW com rodas de tamanho 18
```

## Exercício 1: Criando uma Classe

Crie uma classe chamada `Pessoa` que tenha atributos de nome, idade e cidade. Em seguida, crie um objeto dessa classe e exiba suas informações.

## Exercício 2: Herança e Métodos

Crie uma classe `Animal` com um método `emitir_som()`. Em seguida, crie classes `Cachorro` e `Gato` que herdem da classe `Animal` e implementem o método `emitir_som()` para cada animal fazer seu som característico.

## Exercício 3: Encapsulamento


Crie uma classe `ContaBancaria` com um atributo privado `saldo`. Implemente métodos públicos para depositar, sacar e verificar o saldo.

## Exercício 4: Métodos Especiais

Crie uma classe `Ponto` que represente um ponto no plano cartesiano. Implemente métodos especiais para adição de pontos e exibição do ponto no formato (x, y).

## Exercício 5: Composição

Crie uma classe `Casa` que possua um atributo `cozinha`. A classe `Cozinha` deve ter um método para exibir a mensagem "Preparando o jantar".

## Exercício 6: Agregação

Crie uma classe `Livro` com atributos de título e autor. Em seguida, crie uma classe `Biblioteca` que contenha uma lista de objetos da classe `Livro`.

## Exercício 7: Polimorfismo

Crie uma classe `FormaGeometrica` com um método `calcular_area()`. Implemente classes `Retangulo` e `Circulo` que herdem de `FormaGeometrica` e tenham seus próprios métodos para calcular a área.

## Exercício 8: Reutilização de Código

Reescreva o exercício 2 usando o conceito de herança múltipla para criar um novo animal que combine características de dois animais diferentes.

## Exercício 9: Utilizando Métodos Especiais

Crie uma classe `Data` que represente uma data com atributos de dia, mês e ano. Implemente métodos especiais para comparação de datas.

## Exercício 10: Aplicando os Conceitos

Desenvolva um programa que simule um sistema de gerenciamento de uma loja, utilizando classes como `Produto`, `Cliente`, `CarrinhoDeCompras`, etc., com métodos para adicionar produtos ao carrinho, calcular total, etc.

## Exercício 1: Criando uma Classe

```python
class Pessoa:
 def __init__(self, nome, idade, cidade):
 self.nome = nome
 self.idade = idade
 self.cidade = cidade

# Criando um objeto e exibindo informações
pessoa = Pessoa("Ana", 25, "São Paulo")
print(f"Nome: {pessoa.nome}, Idade: {pessoa.idade}, Cidade: {pessoa.cidade}")
```

## Exercício 2: Herança e Métodos

```python
class Animal:
 def emitir_som(self):
 pass

class Cachorro(Animal):
 def emitir_som(self):
 return "Au Au!"

class Gato(Animal):
 def emitir_som(self):
 return "Miau!"

# Testando os sons
cachorro = Cachorro()
gato = Gato()
print(cachorro.emitir_som()) # Saída: Au Au!
print(gato.emitir_som()) # Saída: Miau!
```

## Exercício 3: Encapsulamento

```python
class ContaBancaria:
 def __init__(self):
 self._saldo = 0

 def deposito(self, valor):
 self._saldo += valor

 def saque(self, valor):
 if valor <= self._saldo:
```

```
 self._saldo -= valor
 else:
 print("Saldo insuficiente")

 def get_saldo(self):
 return self._saldo

# Testando a conta bancária
conta = ContaBancaria()
conta.deposito(100)
conta.saque(30)
print(conta.get_saldo()) # Saída: 70
```

## Exercício 4: Métodos Especiais

```
class Ponto:
 def __init__(self, x, y):
 self.x = x
 self.y = y

 def __str__(self):
 return f"({self.x}, {self.y})"

 def __add__(self, outro_ponto):
 return Ponto(self.x + outro_ponto.x, self.y + outro_ponto.y)

# Testando a adição de pontos
ponto1 = Ponto(1, 2)
ponto2 = Ponto(3, 4)
resultado = ponto1 + ponto2
print(resultado) # Saída: (4, 6)
```

## Exercício 5: Composição

```
class Cozinha:
 def preparar_jantar(self):
 return "Preparando o jantar"

class Casa:
 def __init__(self):
 self.cozinha = Cozinha()
```


```
# Testando a composição
casa = Casa()
print(casa.cozinha.preparar_jantar()) # Saída: Preparando o jantar
```

## Exercício 6: Agregação

```
class Livro:
 def __init__(self, titulo, autor):
 self.titulo = titulo
 self.autor = autor

class Biblioteca:
 def __init__(self):
 self.livros = []

 def adicionar_livro(self, livro):
 self.livros.append(livro)

# Testando a agregação
livro1 = Livro("Python 101", "John Doe")
livro2 = Livro("Data Science Intro", "Jane Smith")

biblioteca = Biblioteca()
biblioteca.adicionar_livro(livro1)
biblioteca.adicionar_livro(livro2)

for livro in biblioteca.livros:
 print(f"Título: {livro.titulo}, Autor: {livro.autor}")
```

# Capítulo 10: Explorando Módulos e Bibliotecas em Python

## Seção 1: Importando Módulos

Importação Simples

Em Python, os módulos são arquivos contendo definições e instruções Python. A importação de um módulo é realizada usando a palavra-chave `import`.

Exemplo:

```
import math

print(math.sqrt(25)) # Saída: 5.0
```

Renomeando Módulos

É possível renomear um módulo ao importá-lo, utilizando a palavra-chave `as`.

Exemplo:

```
import math as m

print(m.sqrt(25)) # Saída: 5.0
```

## Seção 2: Criando e Utilizando Módulos

Criando Módulos

Para criar um módulo em Python, basta criar um arquivo Python (.py) e definir nele as funções ou classes desejadas.

Exemplo:

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

Utilizando Módulos Criados

Após criar um módulo, é possível importá-lo em outros arquivos Python e utilizar suas funcionalidades.

Exemplo:

```
import my_module

print(my_module.saudacao("Alice")) # Saída: Olá, Alice!
```

## Seção 3: Bibliotecas Populares em Python

Módulo os

O módulo `os` fornece funcionalidades relacionadas ao sistema operacional, como manipulação de arquivos e diretórios.

Exemplo:

```
import os

print(os.getcwd()) # Retorna o diretório de trabalho atual
```

Módulo random

O módulo `random` é utilizado para gerar números aleatórios.

Exemplo:

```
import random

print(random.randint(1, 10)) # Retorna um número aleatório entre 1 e 10
```

Módulo math

O módulo `math` oferece funções matemáticas.

Exemplo:

```
import math

print(math.pi) # Retorna o valor de pi
print(math.sqrt(16)) # Retorna a raiz quadrada de 16
```

Explorar módulos e bibliotecas é fundamental para expandir as capacidades do Python. Com essas ferramentas, é possível realizar tarefas complexas de forma mais eficiente e poderosa.

# Capítulo 11: Utilizando MQTT em Python


# Seção 1: Instalando a Biblioteca paho-mqtt

Para começar, é necessário instalar a biblioteca `paho-mqtt`:

# Seção 2: Conectando-se ao Broker MQTT

Importando a Biblioteca

```
import paho.mqtt.client as mqtt
```

Criando o Cliente MQTT

```
# Callback quando o cliente se conecta ao broker
def on_connect(client, userdata, flags, rc):
 print(f"Conectado ao broker com código de resultado: {rc}")
 client.subscribe("topico/teste") # Subscrevendo ao tópico de teste

# Callback quando uma nova mensagem é recebida
def on_message(client, userdata, msg):
 print(f"Mensagem recebida no tópico {msg.topic}: {msg.payload.decode()}")

# Configurando o cliente MQTT
client = mqtt.Client()

# Associando os callbacks ao cliente
client.on_connect = on_connect
client.on_message = on_message

# Conectando ao broker MQTT
client.connect("test.mosquitto.org", 1883, 60) # Conexão ao broker MQTT público
```

# Seção 3: Publicando e Subscrevendo a Tópicos MQTT

Publicando Mensagens

```
client.publish("topico/teste", "Olá, MQTT!")
```

```
# Mantendo a conexão e recebendo mensagens indefinidamente

client.loop_forever()
```

## Seção 4: Finalizando a Conexão

```
client.disconnect()
```

Esses são os passos básicos para utilizar o MQTT em Python com a biblioteca `paho-mqtt` e se conectar ao broker test.mosquitto.org.

programa que cria um formato de chat usando MQTT, onde os participantes se comunicam em diferentes tópicos baseados nos seus nomes:

```
import paho.mqtt.client as mqtt

# Callback quando o cliente se conecta ao broker
def on_connect(client, userdata, flags, rc):
 print(f"Conectado ao broker com código de resultado: {rc}")
 client.subscribe("aula_mqtt/python/mensagens/#") # Subscrevendo a todos os
tópicos de mensagens

# Callback quando uma nova mensagem é recebida
def on_message(client, userdata, msg):
 print(f"Mensagem recebida no tópico {msg.topic}: {msg.payload.decode()}")

# Configurando o cliente MQTT
client = mqtt.Client()

# Associando os callbacks ao cliente
client.on_connect = on_connect
client.on_message = on_message

# Conectando ao broker MQTT
client.connect("test.mosquitto.org", 1883, 60) # Conexão ao broker MQTT público

# Função para enviar mensagem
def enviar_mensagem():
 nome = input("Digite seu nome: ")
 while True:
 mensagem = input("Digite sua mensagem (ou 'sair' para sair): ")
 if mensagem.lower() == 'sair':
 break
 # Publicando a mensagem no tópico correspondente ao nome
```

```
 client.publish(f"aula_mqtt/python/mensagens/{nome}", mensagem)

# Iniciando a função para enviar mensagens em uma thread separada
import threading
thread = threading.Thread(target=enviar_mensagem)
thread.start()

# Mantendo a conexão e recebendo mensagens indefinidamente
client.loop_forever()
```

Este programa cria um chat simples usando MQTT. Os participantes se conectam ao tópico base `aula_mqtt/python/mensagens/` seguido pelo nome da pessoa para enviar e receber mensagens. Cada participante pode enviar mensagens utilizando seu nome como parte do tópico.

## Exercícios sobre Bibliotecas em Python:

1) Importando e Utilizando Módulos:
    a) Escreva um programa que utilize o módulo `random` para gerar 5 números aleatórios entre 1 e 20.
2) Explorando Funcionalidades da Biblioteca `os`:
    a) Crie uma função que liste todos os arquivos de um diretório específico usando o módulo `os`.
3) Explorando o Módulo `math`:
    a) Implemente uma função que calcule a área de um círculo com base no raio fornecido pelo usuário utilizando o módulo `math`.
4) Criando seu Próprio Módulo:
    a) Crie um módulo Python que contenha uma função para calcular a média de uma lista de números.

## Exercício 1: Importando e Utilizando Módulos

```
import random

for _ in range(5):
 print(random.randint(1, 20))
```

## Exercício 2: Explorando Funcionalidades da Biblioteca `os`

```
import os
```

```python
def listar_arquivos(diretorio):
 for root, dirs, files in os.walk(diretorio):
 for file in files:
 print(os.path.join(root, file))

listar_arquivos('/caminho/do/diretorio')
```

## Exercício 3: Explorando o Módulo `math`

```python
import math

def calcular_area_circulo(raio):
 return math.pi * raio ** 2

raio = float(input("Digite o raio do círculo: "))
print(f"A área do círculo é: {calcular_area_circulo(raio)}")
```

## Exercício 4: Criando seu Próprio Módulo

```python
def calcular_media(lista):
 return sum(lista) / len(lista)
```

No script principal:

```python
from meu_modulo import calcular_media

numeros = [1, 2, 3, 4, 5]
print(f"A média dos números é: {calcular_media(numeros)}")
```