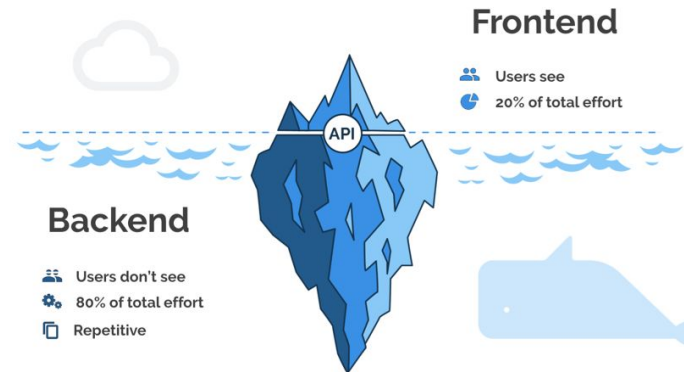
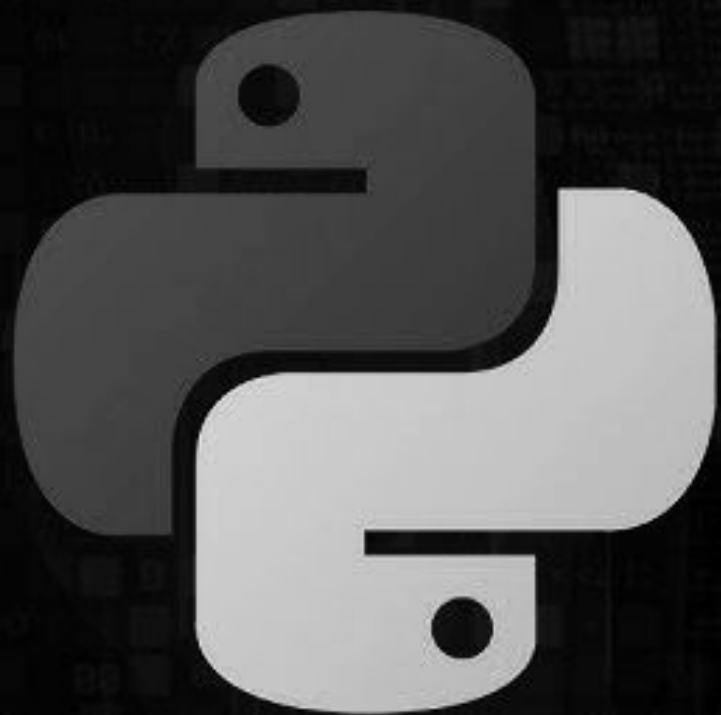


## Objetivos da Aula:

- Conhecer funções
- Entender o conceito de módulos em Python.
- Aprender como criar, usar e importar módulos.
- Introdução aos Pacotes.
- Identificar casos de uso comuns para módulos.





# FUNÇÕES EM PYTHON

## O que é uma Função?

uma função é um **bloco de código** que pode ser reutilizado, projetado para realizar uma tarefa específica. Analogia: Uma "caixa" que recebe entradas "parâmetros" e realiza um processamento, e pode retornar uma saída.

```
def nome_da_funcao(parametro1, parametro2):  
    valor = parametro1 + parametro2  
    return valor
```

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

## Parâmetros e Argumentos

**parâmetros** (definidos na função) e **argumentos** (passados na chamada da função).

Tipos de parâmetros:

**obrigatórios** e **opcionais**.

```
def saudar(nome, saudacao="Olá"):  
    return f"{saudacao} ,  
    {nome} !"  
  
print(saudar("Pedro"))  
print(saudar("Pedro", "Bom dia"))
```

## Funções com Múltiplos Parâmetros

Usando parâmetros arbitrários com `*args` e `**kwargs`:

- **`*args`**: Para passar múltiplos argumentos **posicionais** são acessados como lista ou tupla.

```
def soma_total(*args):  
    return args[1]  
  
print(soma_total(1, 2, 3, 4))
```

## Funções com Múltiplos Parâmetros

Usando parâmetros arbitrários com `*args` e `**kwargs`:

- **`**kwargs`**: Para passar múltiplos argumentos **nomeados** são acessados como um dicionário.

```
def detalhes_usuario(**kwargs):  
    for chave, valor in kwargs.items():  
        print(f"{chave}: {valor}")  
  
detalhes_usuario(nome="João", idade=30,  
                 cidade="São Paulo")
```

## Funções Aninhadas (nested functions) e closures

### Situações para Usar Funções dentro de Funções:

1. **Encapsulamento de lógica específica:** Quando você tem uma lógica que só faz sentido dentro do contexto de outra função e não precisa estar disponível fora dessa função.
2. **Evitar repetição de código:** Se um pedaço de lógica precisa ser reutilizado em diferentes partes de uma função maior, mas não é necessário em outros lugares do código.

```
def externo():  
    contador = 0  
  
    def interno():  
        nonlocal contador  
        contador += 1  
  
    interno()  
    interno()  
    return contador  
  
print(externo())
```

## Funções Aninhadas (nested functions) e closures

```
def autenticar(usuario, senha):  
    def validar_credenciais(usuario, senha):  
        return usuario == "admin" and senha == "1234"  
  
    if validar_credenciais(usuario, senha):  
        return "Acesso permitido"  
    else:  
        return "Acesso negado"  
  
print(autenticar("admin", "1234"))
```



## Funções Aninhadas (nested functions) e closures

```
def saudacao(nome):  
    def adicionar_prefixo():  
        return "Olá"  
  
    saudacao_completa = f"{adicionar_prefixo()}, {nome}!"  
    return saudacao_completa  
  
print(saudacao("Maria"))
```

## Funções como objetos de primeira classe

Em Python, funções são “cidadãs de primeira classe” (first-class citizens). Isso significa que você pode:

- Atribuir funções a variáveis.
- Passar funções como argumento para outras funções.
- Retornar funções de dentro de outras funções.

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

```
somar = somar_dois_numeros  
resultado = somar(10, 5)  
print(resultado)
```

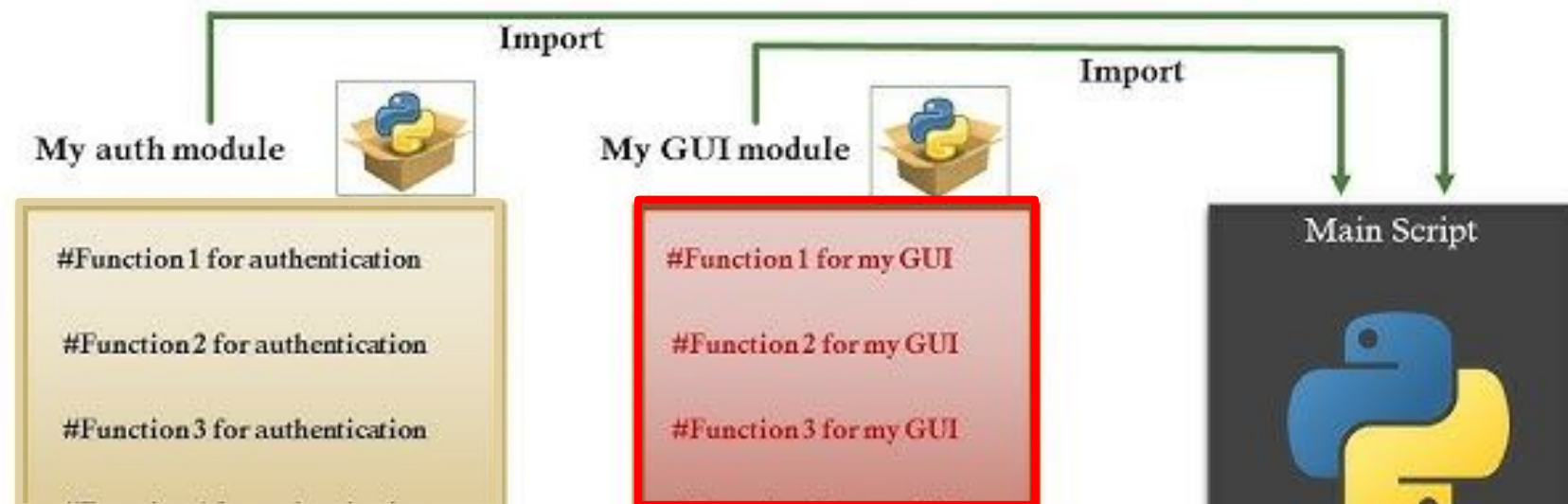
## Funções lambda (funções anônimas)

São funções rápidas, de uma única expressão, muito úteis para criar funções simples que serão usadas uma única vez, por exemplo, como saber se um número é par

```
quadrado = lambda x: x ** 2  
print(quadrado(4))
```

```
numero_par = lambda x: x % 2 == 0  
print(numero_par(4))
```

# Python Modules



## Módulos em Python

Em Python, um módulo é um arquivo contendo definições e instruções Python.

Os módulos podem definir funções, classes e variáveis que podem ser reutilizadas em outros programas Python.

calculos.py

```
def somar( x, y):  
    return x + y
```

main.py

```
import calculos
```

```
resultado = calculos.somar(3, 8)  
print(resultado)
```

# Formas de importar módulos

## 1ª forma



calculadora.py &gt; adicionar

```

1  def subtrair(a, b):
2      return a - b
3
4  def multiplicar(a, b):
5      return a * b
6
7  def dividir(a, b):
8      if b == 0:
9          return "Erro: Divisão por zero não é permitida."
10         return a / b
11
12
13 def adicionar(a, b):
14     return a + b

```

```

1  import calculadora
2
3
4  def main():
5      escolha=5
6      while escolha != 0:
7          print("\n== Calculadora Básica ==\n")
8
9          a = float(input("Digite o primeiro número: "))
10         b = float(input("Digite o segundo número: "))
11

```

```

16     DIVISÃO )
17
18     : input("\nEscolha a operação (1/2/3/4) ou 0")
19
20     :a == '1':
21         :tado = calculadora.adicionar(a, b)
22         :icao = "Adição"
23     :lha == '2':
24         :tado = calculadora.subtrair(a, b)
25         :icao = "Subtração"
26     :lha == '3':
27         :tado = calculadora.multiplicar(a, b)
28     :a == '4':
29         :tado = calculadora.dividir(a, b)
30

```

# Formas de importar módulos

2ª forma



```
You, 24 seconds ago | 1 author (You)
from calculadora import adicionar, subtrair, multiplicar, dividir
```

```
def main():      You, 8 minutes ago • pacotes em python
```

```
    escolha=5
```

```
    while escolha != 0:
```

```
        print("\n=== Calculadora Básica ===\n")
```

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

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

```
        print("\nOperações Disponíveis:")
```

```
        print("1. Adição")
```

```
        print("2. Subtração")
```

```
        print("3. Multiplicação")
```

```
        print("4. Divisão")
```

```
calculadora.py > adicionar
```

```
1 def subtrair(a, b):
```

```
    return a - b
```

```
3
```

```
4 def multiplicar(a, b):
```

```
    return a * b
```

```
6
```

```
7 def dividir(a, b):
```

```
    if b == 0:
```

```
        return "Erro: Divisão por zero não é permitida."
```

```
    return a / b
```

```
10
```

```
11
```

```
12
```

```
13 def adicionar(a, b):
```

```
    return a + b
```

```
14
```

```
main.py > main
```

```
18 escolha = input("\nEscolha a operação (1/
```

```
19
```

```
20 if escolha == '1':
```

```
    resultado = adicionar(a, b)
```

```
    operacao = "Adição"
```

```
23 elif escolha == '2':
```

```
    resultado = subtrair(a, b)
```

```
    operacao = "Subtração"
```

```
26 elif escolha == '3':
```

```
    resultado = multiplicar(a, b)
```

```
    operacao = "Multiplicação"
```

```
29 elif escolha == '4':
```

```
    resultado = dividir(a, b)
```

```
    operacao = "Divisão"
```

```
32 else:
```

```
    print("Saindo....")
```

```
    return
```

## Para que servem os Módulos?

Os módulos servem para organizar o código Python em unidades lógicas e reutilizáveis. **Eles facilitam a manutenção, colaboração** e escalabilidade de projetos Python, permitindo a separação de funcionalidades relacionadas em arquivos diferentes.

```
lista_palavras = ["cachorro", "gato",  
"pássaro", "elefante", "gato", "cobra"]
```

```
blacklist = ["pássaro", "cobra"]
```

```
for palavra in blacklist:  
    while palavra in lista_palavras:  
        lista_palavras.remove(palavra)
```

```
print(lista_palavras)
```



# Como Criar um Módulo em Python?

```
# lista.py
```

```
lista_palavras = ["cachorro", "gato",  
"pássaro", "elefante", "gato", "cobra"]
```

```
# blacklist.py
```

```
blacklist = ["gato", "cobra"]
```

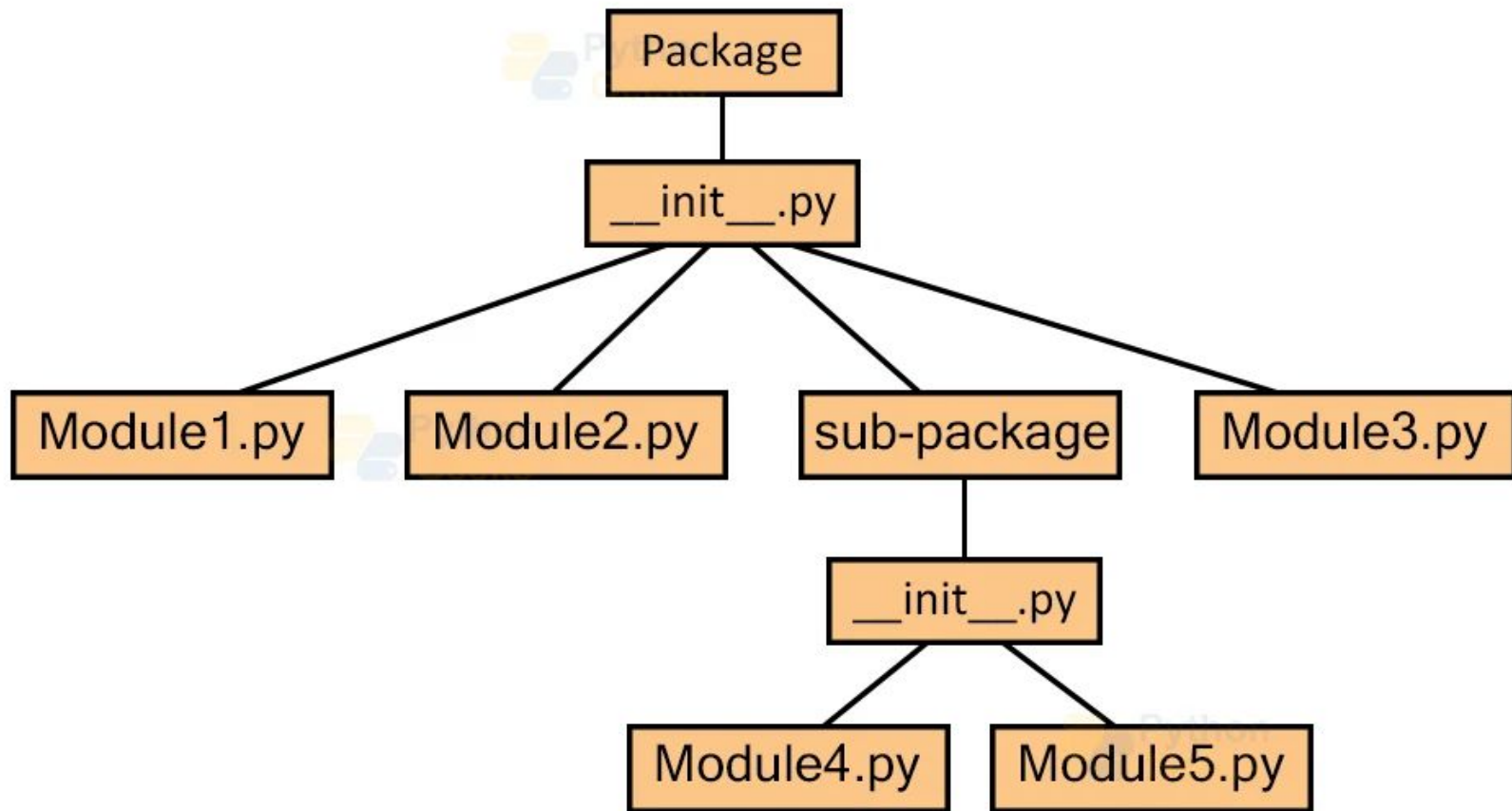
```
# remover.py
```

```
def remover_palavras(lista, blacklist):  
    for palavra in blacklist:  
        while palavra in lista:  
            lista.remove(palavra)  
    return lista
```

```
# main.py
```

```
import lista as ls  
import blacklist  
from remover import remover_palavras  
  
remover_palavras(ls.lista_palavras,  
blacklist.blacklist)  
  
print(ls.lista_palavras)
```

# Structure of Packages



## O que é um Pacote?


Um **pacote** em Python é uma coleção de módulos organizados em uma estrutura de diretórios hierárquica. Ele permite agrupar módulos relacionados, facilitando a organização e reutilização do código.

> calculadora

 main.py

▼ calculadora

 \_\_init\_\_.py

 adicao.py

 divisao.py

 multiplicacao.py

 subtracao.py

 main.py

# \_\_init\_\_.py

## ✓ PYTHON\_PACOTES

> .git

✓ calculadora

🔗 \_\_init\_\_.py

🔗 adicao.py

🔗 divisao.py

🔗 multiplicacao.py

🔗 subtracao.py

calculadora > 🔗 \_\_init\_\_.py > ...

```

1  from .adicao import adicionar
2  from .subtracao import subtrair
3  from .multiplicacao import multiplicar
4  from .divisao import dividir
5
6  __all__ = ['adicionar', 'subtrair', 'multiplicar', 'dividir']
7  |

```

calculadora > 🔗 adicao.py > 📦 adicao

```

1  def adicionar(a, b):
2  |     return a + b

```

calculadora > 🔗 subtracao.py > 📦 subtracao

```

1  def subtrair(a, b):
2  |     return a - b

```

calculadora > 🔗 multiplicacao.py > 📦 multiplicacao

```

1  def multiplicar(a, b):
2  |     return a * b

```

# \_\_init\_\_.py

## PYTHON\_PACOTES

&gt; .git

- calculadora

- \_\_init\_\_.py

- adicao.py

- divisao.py

- multiplicacao.py

- subtracao.py



calculadora >  \_\_init\_\_.py > ...

```

1  from .adicao import adicionar
2  from .subtracao import subtrair
3  from .multiplicacao import multiplicar
4  from .divisao import dividir
5
6  __all__ = ['adicionar', 'subtrair', 'multiplicar', 'dividir']
7  |

```

## adicao.py

calculadora >  adicao.py >  adicao

```

1  def adicionar(a, b):
2  |     return a + b

```

## subtracao.py

calculadora >  subtracao.py >  subtr

```

1  def subtrair(a, b):
2  |     return a - b

```

## multiplicacao.py

calculadora >  multiplicacao.py >  mult


```

1  def multiplicar(a, b):
2  |     return a * b

```

# main.py

> calculadora

 main.py

```
1  from calculadora import adicionar, subtrair
2
3  result1 = adicionar(2,3)
4  result2 = subtrair(2,3)
5
6  print(f'soma 2 e 3 : {result1} \nsubtrai 2 e 3: {result2}')
```

## Acessando Bibliotecas Externas em Python

Além de criar seus próprios módulos, Python possui uma vasta coleção de bibliotecas externas que podem ser importadas e utilizadas em seus projetos.

Por exemplo, para usar a biblioteca `random` para gerar números aleatórios:

```
import random
```

```
numero_aleatorio = random.randint(1,  
100)
```

```
print("Número aleatório:",  
numero_aleatorio)
```

## Casos de Uso Comuns para Módulos

- Reutilização de código: Módulos permitem que você reutilize funções, classes e variáveis em vários programas.
- Organização do código: Módulos ajudam a organizar o código em unidades lógicas, facilitando a compreensão e manutenção do código.
- Colaboração: Módulos permitem que equipes de desenvolvimento trabalhem em partes separadas de um projeto de forma independente.
- Utilização de bibliotecas externas: Importar bibliotecas externas permite acessar funcionalidades avançadas que não estão disponíveis na biblioteca padrão de Python.



## Atividade Prática 2 Criando um Módulo de Contagem de Palavras

Crie o Módulo `contagem_palavras.py`:

- Defina uma função chamada `contar_palavras` que receba um texto como parâmetro.

Implemente a Função `contar_palavras`:

- Converter para Minúsculas: Use `lower()` para transformar todo o texto em minúsculas.
- Dividir em Palavras: Utilize `split()` para separar o texto em uma lista de palavras.
- Contar Palavras:
  - Inicialize um dicionário vazio para armazenar a contagem.
  - Percorra a lista de palavras e incremente a contagem de cada palavra no dicionário.
- Retornar o Dicionário: Após a contagem, retorne o dicionário com as palavras e suas respectivas frequências.

## Atividade Prática a variável texto

o texto a ser utilizado na próxima atividade.

```
texto = "Python é uma linguagem de programação de alto nível, interpretada e de propósito  
geral. Criada por Guido van Rossum e lançada pela primeira vez em 1991, a filosofia do  
Python é enfatizar a legibilidade do código com sua sintaxe simples e fácil de aprender. A  
linguagem foi projetada com a filosofia de enfatizar a importância do esforço do programador  
sobre o esforço computacional. Prioriza a legibilidade do código sobre a velocidade ou  
expressividade. Combina uma sintaxe concisa e clara com os recursos poderosos de sua  
biblioteca padrão e por módulos e frameworks desenvolvidos por terceiros. Python é uma  
linguagem de propósito geral de alto nível, multiparadigma, suporta o paradigma orientado a  
objetos, imperativo, funcional e procedural. Possui tipagem dinâmica e uma de suas  
principais características é permitir a fácil leitura do código e exigir poucas linhas de código  
se comparado ao mesmo programa em outras linguagens.
```

"

## Exercício.

Criar uma calculadora.

- Função específica para somar
- Função específica para multiplicar
- Função específica para dividir
- Função específica para subtrair
- Função específica para exibir o resultado

### Exercício 1: Função de Verificação de Número Primo

- Criar uma função que receba um número e retorne **True** se o número for primo, e **False** se não for.

### Exercício 2: Função Recursiva para Somar uma Lista

- Criar uma função recursiva que calcule a soma de todos os elementos de uma lista.

### Exercício 3: Função para Contagem de Palavras

- Escrever uma função que receba uma string e conte o número de palavras.

### Exercício 4: Função com **\*args** e **\*\*kwargs** para Registrar Várias Informações

- Criar uma função que aceite um número indeterminado de argumentos e informações extras nomeadas, e exiba-as.

## Exercícios: Listas, Loops e Funções

### Calcular a Média de Notas dos Alunos

#### Descrição:

1. **Passo 1:** Crie uma lista com as notas de cinco alunos.
2. **Passo 2:** Use um loop foreach para calcular a soma das notas.
3. **Passo 3:** Crie uma função que recebe a soma das notas e o número de alunos, e retorna a média.
4. **Passo 4:** Exiba a média das notas na tela.

## Exercícios: lista, Loops e Funções em python

### Filtrar Números Pares de um **lista**

**Passo 1:** Crie um lista com dez números inteiros.

**Passo 2:** Crie uma função que verifica se um número é par.

**Passo 3:** Use um loop for para iterar sobre a lista e usar a função de verificação de números pares.

**Passo 4:** Armazene os números pares em um novo lista e exiba-os na tela.

## Referências

BEAZLEY D.; JONES, B.K. Python Cookbook: Receitas para dominar Python. 3 ed. São Paulo: Novatec, 2019.

CRUZ, FJ. Python: escreva seus primeiro programas. 1.ed. São Paulo: Casa do Código, 2021.

PEREIRA, E, DOUGLAS MICHAEL. Trilhas Python: Programação multiparadigma e desenvolvimento web com Flask. 1.ed. São Paulo: Casa do Código, 2020.

