

A decorative graphic on the left side of the slide. It consists of a blue parallelogram and a light green parallelogram, both tilted at an angle. The blue shape is in the foreground, and the green shape is partially behind it. They are set against a dark blue background with faint, lighter blue diagonal stripes.

Introdução a Python



Quem sou eu?

- Caio Cézar Salomão Andrade
- Aluno Bacharelado em Sistemas de Informação
- Estagiário na empresa Aplicativo Preço do Gás
- Voluntário no projeto IPCB em cooperação com o campus IFMG Bambuí



Quem sou eu?

- Erick Sena Godinho aka Melhor Programador de MG
- Técnico em Informática pelo IFMG - SJE
- Aluno Bacharelado em Sistemas de Informação
- Monitor da Disciplina de Introdução à Programação



Pré-requisitos

- Visual Studio Code, PyCharm ou Google Colab
- Python

Link para o repositório: github.com/caioandrade42/



Extensões Recomendadas(VS Code)

- Python
- Pylance
- Python indent



Estruturas básicas

- Operadores lógicos
- Operadores aritméticos
- Estruturas de comparação
- Estruturas de repetição
- Declaração de uma função



Tipos de operadores lógicos



```
saldo = 1000
saque = 200
limite = 100

# Operador AND
print(saldo >= saque and saque <= limite)

# Operador OR
print(saldo >= saque or saque <= limite)

# Operador NOT
print(not saldo >= saque)
```

Operadores de identidade e associação



```
curso = "Curso de Python"
nome_curso = curso
linguagens=["Java","Python","C++","C#","C"]
saques=[150,300,655,1500]
```

```
# Operador de identidade
print(nome_curso is curso)
print(saldo is limite)
```

```
# Operadores de associacao
print("Python" in nome_curso)
print("python" in nome_curso)
print("C#" in linguagens)
print("20" in saques)
```




Operadores aritméticos

```
# Divisao
print(10 / 4)

# Divisao inteira
print(10 // 4)

# Modulo(resto da divisao)
print(10 % 4)

# Multiplicacao
print(2 * 2)

# Exponenciacao
print(2 ** 3)
```



Declarando uma função



```
def nome_funcao(parametro: str) -> str:  
    return parametro + "!"
```

Estruturas de comparação



```
# Condicionais
valor = 5
if valor > 0:
    print("Positivo")
elif valor < 0:
    print("Negativo")
else:
    print("Nulo")
```



```
caso = 5
if caso == 1:
    print("caso 1")
elif caso == 2:
    print("caso 2")
else:
    print("caso padrao")
```

Estruturas de repetição



```
for i in range(10):  
    print(i)  
  
for i in range(0, 10, 2):  
    print(i)  
  
array = [1, 3, 5]  
for i in array:  
    print(i)  
  
for i in range(len(array)): # for i in range(3)  
    print(i)  
  
for i in "Curso Python":  
    print(i)
```



```
i = 0  
while i < 10:  
    i += 1  
    print(i)  
  
i = 0  
while True:  
    i += 1  
    if i == 10: # Condição de Parada  
        break
```



Manipulação de Strings

- Manipulações simples
- Fatiamento de Strings
- Interpolação de Strings
- Strings com múltiplas linhas

Manipulações simples



```
PI = 3.14159
```

```
print(f"Valor de PI: {PI:.2f}")
```

```
print(f"Valor de PI: {PI:.4f}")
```

```
print("{:<30}".format("alinhado na esquerda"))
```

```
print("{:>30}".format("alinhado na direita"))
```

```
print("{:^30}".format("centralizado"))
```

```
print("{:#^30}".format("centralizado"))
```

```
print("Python".center(14, "#"))
```

Manipulações simples

```
# Sempre mostra o sinal
print("{:+f}; {:+f}".format(3.14, -3.14))

# Caso o numero for positivo, mostra com um espaço na frente
print("{: f}; {: f}".format(3.14, -3.14))

# Mostra o sinal apenas de for negativo (mesmo que {:f})
print("{:-f}; {:-f}".format(3.14, -3.14))

# Mostra o valor em bases diferentes
print("int: {0:d};  hex: {0:x};  oct: {0:o};  bin: {0:b}".format(42))
```

```
curso = "  Hello World      "
print(curso + "  .")
print(curso.lstrip() + ".")
print(curso.rstrip() + ".")
print(curso.strip() + ".")
print("+".join(curso.strip()))
```

Fatiamento de Strings



```
## Imprime o valor na posicao 0 da string
print(nome[0])

## Imprime o conteudo da string ate o indice 4
print(nome[:4])

## Imprime o conteudo da string a partir da posicao 11
print(nome[11:])

## Imprime o conteudo da string da posicao 6 a 10
print(nome[5:10])

## Imprime em 'step' dois a dois
print(nome[5:10:2])
```



```
## Imprime toda a string
print(nome[:])

## Inverte a string
print(nome[::-1])

## Imprime do 7 caracter (contando a
partir do fim) até o final
print(nome[-7:])

# Separando a frase em espacos
print(nome.split())

# Separando a frase a cada letra 'a'
print(nome.split('a'))
```


Interpolação de Strings



```
nome = "Caio"  
idade = 28  
profissao = "Programador"  
linguagem = "Python"
```

```
dados = {"nome": "Caio", "idade": 28,  
         "linguagem": "Python", "profissao": "Programador"}
```

```
print("1 - Ola me chamo %s. Tenho %d anos de idade, trabalho como %s e estou matriculado "  
      "no curso de %s" % (nome, idade, profissao, linguagem))  
print("2 - Ola me chamo {nome}. Tenho {idade} anos de idade, trabalho como {profissao} e estou matriculado "  
      "no curso de {linguagem}".format(linguagem=linguagem, profissao=profissao, idade=idade, nome=nome))  
print("3 - Ola me chamo {}. Tenho {} anos de idade, trabalho como {} e estou matriculado "  
      "no curso de {}".format(nome, idade, profissao, linguagem))  
print(f"4 - Ola me chamo {nome}. Tenho {idade} anos de idade, trabalho como {profissao} e estou matriculado "  
      f"no curso de {linguagem}.")  
print("5 - Ola me chamo {nome}. Tenho {idade} anos de idade, trabalho como {profissao} e estou matriculado "  
      "no curso de {linguagem}.".format(**dados))
```

Strings com múltiplas linhas



```
nome = "Caio"
```

```
mensagem = f"""  
Ola meu nome e {nome},  
Eu estou aprendendo Python  
"""
```

```
print(mensagem)
```

```
mensagem_aspas_simples = f'''  
Ola meu nome e {nome},  
Eu estou aprendendo Python  
'''
```

```
print(mensagem_aspas_simples)
```



Estruturas de Dados

- Array
- List
 - Stack
 - Queue
- Tuple
- Set
- Dictionary

Array



```
import array as arr

# Criar um array de inteiros
array = arr.array('i', [1, 2, 3, 1])

# Acessar valor no array
print(array[1])

# Slice array
print(array[1:2])

# Procurar valor no array
print(array.index(1))

# Contar quantas vezes um valor apareceu
print(array.count(1))
```

List



```
# Criando uma lista de animais
animais = ["cachorro", "gato", "coelho", "papagaio", "gato", "coelho"]

# Contando quantos gatos há na lista
animais.count("gato")
#2

# Contando quantos leões há na lista
animais.count("leão")
#0

# Encontrando o índice do primeiro coelho na lista
animais.index("coelho")
#2

# Encontrando o índice do próximo coelho a partir da posição 3
animais.index("coelho", 3)
#5
```

List




```
# Adicionando um elefante à lista
animais.append("elefante")
#animais ["coelho", "gato", "papagaio", "coelho", "gato", "cachorro", "elefante"]

# Removendo o último elemento da lista
animais.pop()
#"papagaio"

# Invertendo a ordem dos elementos da lista
animais.reverse()
#animais ["coelho", "gato", "papagaio", "coelho", "gato", "cachorro"]

# Ordenando a lista em ordem alfabética
animais.sort()
#animais ["cachorro", "coelho", "coelho", "elefante", "gato", "gato", "papagaio"]
```

List como Stack



```
stack = [3, 4, 5]
stack.append(6)
stack.append(7)
#stack [3, 4, 5, 6, 7]

stack.pop()
#7
#stack [3, 4, 5, 6]

stack.pop()
#6

stack.pop()
#5
#stack [3, 4]
```

List como Queue



```
# Importando a classe deque do módulo collections
from collections import deque

# Criando uma lista de frutas usando deque
frutas = deque(["maçã", "banana", "laranja"])

frutas.append("uva")
frutas.append("abacaxi")

# Removendo o primeiro elemento da lista
frutas.popleft() # O primeiro a chegar agora sai
#"maçã"

# Removendo o segundo elemento da lista
frutas.popleft() # O segundo a chegar agora sai
#"banana"

# Imprimindo a lista restante em ordem de chegada
#frutas deque(["laranja", "uva", "abacaxi"])
```


Tuple

```
t = 12345, 54321, 'hello!'
# t[0] -> 12345
# t -> (12345, 54321, 'hello!')
```

Tuples podem ser aninhados:

```
u = t, (1, 2, 3, 4, 5)
# u -> ((12345, 54321, 'hello!'), (1, 2, 3, 4, 5))
```

Tuples são imutáveis:

```
t[0] = 88888
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

Mas elas podem conter objetos mutáveis:

```
v = ([1, 2, 3], [3, 2, 1])
# v -> ([1, 2, 3], [3, 2, 1])
```

```
empty = ()
singleton = 'hello',    # <-- observe a virgula
```

```
x, y, z = t
```

Set

```
cesta = {'maçã', 'laranja', 'maçã', 'pera', 'laranja', 'banana'}
set_vazio = set() # não se cria set vazio com {}

print(cesta)
# mostra que as duplicatas foram removidas {'laranja', 'banana', 'pera', 'maçã'}

'laranja' in cesta
# teste de pertinência rápido
# True
'grama de caranguejo' in cesta
# False

Demonstra operações de conjunto em letras únicas de duas palavras
a = set('abracadabra') b = set('alacazam')
```

a	# letras unicas em a
# {'a', 'r', 'b', 'c', 'd'}	
a - b	# letras em a, mas não em b
# {'r', 'd', 'b'}	
a b	# letras em a ou b ou ambos
# {'a', 'c', 'r', 'd', 'b', 'm', 'z', 'l'}	
a & b	# letras em a e b
# {'a', 'c'}	
a ^ b	# letras em a ou b, mas nao em ambos
# {'r', 'd', 'b', 'm', 'z', 'l'}	

Dictionary



```
nomes = {'ana': 1234, 'bia': 5678, 'carla': 9012}
nomes['dani'] = 3456

# nomes {'ana': 1234, 'bia': 5678, 'carla': 9012, 'dani': 3456}
# nomes['bia'] 5678

del nomes['carla']
nomes['elisa'] = 7890
# nomes {'ana': 1234, 'bia': 5678, 'dani': 3456, 'elisa': 7890}

list(nomes)
# ['ana', 'bia', 'dani', 'elisa']

sorted(nomes)
# ['ana', 'bia', 'dani', 'elisa']

'dani' in nomes
# True

'ana' not in nomes
# False

knights = {'gallahad': 'the pure', 'robin': 'the brave'}
for k, v in knights.items():
    print(k, v)
```

Orientação a Objetos

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

    def __str__(self):
        return f"Nome: {self.nome}, Idade: {self.idade}, Peso: {self.peso}"

class Aluno(Pessoa):
    def __init__(self, matricula, nome, idade, peso):
        self.__matricula = matricula
        super().__init__(nome, idade, peso)

    def __str__(self):
        return f"Matricula: {self.__matricula}, Nome: {self.nome}, Idade: {self.idade}, Peso: {self.peso}"

pessoa = Pessoa("Pessoa", 35, 75)
aluno = Aluno(1234, "Aluno", 19, 70)
print(pessoa)
print(aluno)
```



Referências

<https://docs.python.org/3/>

<https://docs.python.org/3/tutorial/datastructures>