



# Paradigmas de linguagens de programação em python

Professores:

Sebastião Rogerio feat. Kayo Monteiro

# Agenda

01

Introdução a  
Python

02

Variáveis

03

Primeiros Trabalhos

04

Estruturas



# 01

## Introdução a Python:

# Python

Python é uma linguagem de programação relativamente simples que foi criada por Guido van Rossum em 1991, ela é de **alto nível**, **interpretada** e de **alta produtividade** (imperativa, Orientada Objeto e Funcional)



# Python – Princípios

- Não há declaração de tipos de variáveis
- Não há Begin e End, { } ou ;
- Comentários são feitos com #
- Comentários de mais de uma linha """
- Identação é **OBRIGATÓRIA**
- Organização é fundamental
- É case-sensitive



02

Variáveis

```
>>> a = 10
>>> a
10
```

```
>>> b = 1.2
>>> b
1.2
```

```
>>> c = "Olá Mundo"
>>> c
'Olá Mundo'
```

# Variáveis

- São pequenos espaços de memória, utilizados para armazenar e manipular dados;
- Em Python, os tipos de dados básicos são: tipo inteiro, float e tipo string;
- Cada variável pode armazenar apenas um tipo de dado a cada instante;
- Em Python, diferentemente de outras linguagens de programação, não é preciso declarar de que tipo será cada variável no início do programa.

# Variáveis

- A atribuição de valor para uma variável pode ser feita utilizando o comando `input()`, que solicita ao usuário o valor a ser atribuído à variável.



```
>>> nome = input("Entre com o seu nome: ")
Entre com o seu nome: Fulano da Silva
>>> nome
'Fulano da Silva'
```



# Variáveis

O comando `input()`, sempre vai retornar uma `string`. Nesse caso, para retornar dados do tipo inteiro ou `float`, é preciso converter o tipo do valor lido. Para isso, utiliza-se o `int (string)` para converter para o tipo inteiro, ou `float (string)` para converter para o tipo `float`.



```
>>> num = int(input("Entre com um numero? :"))
Entre com um numero? :100
>>> num
100
```


# Variáveis

- Em Python, os nomes das variáveis devem ser iniciados com uma letra, mas podem possuir outros tipos de caracteres, como números e símbolos;
- O símbolo sublinha ( `_` ) também é aceito no início de nomes de variáveis;

# Exemplo de nomes Variáveis

Nome	Válido	Comentários
a3	Sim	Embora contenha um número, o nome a3 inicia com letra.
velocidade	Sim	Nome formado com letras.
velocidade90	Sim	Nome formado por letras e números, mas inicia com letras.
salario_médio	Sim	O símbolo ( _ ) é permitido e facilita a leitura de nomes grandes.
salario médio	Não	Nomes de variáveis não podem conter espaços em branco.
_salário	Sim	O sublinha ( _ ) é aceito em nomes de variáveis, mesmo no início.
5A	Não	Nomes de variáveis não podem começar com números

# Strings


A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. It displays a Python command and its output.

```
>>> print("Olá mundo")  
Olá mundo  
>>>
```

- Uma string é uma sequência de caracteres simples. Na linguagem Python, as strings são utilizadas com aspas simples ('... ') ou aspas duplas ("...");
- Para exibir uma string, utiliza-se o comando print();

# Strings

- Para concatenar strings, utiliza-se o operador +



```
>>> print("Apostila"+"Python")
ApostilaPython
>>> a='Programação'
>>> b='Python'
>>> c=a+b
>>> print(c)
ProgramaçãoPython
```

# Strings

Em Python, existem várias funções (métodos) para manipular strings. Na tabela a seguir são apresentados os principais métodos para a manipulação as strings.

# Exemplo de manipulação de strings

Método	Descrição	Exemplo
len()	Retorna o tamanho da string.	teste = "Apostila de Python" len(teste) 18
capitalize()	Retorna a string com a primeira letra maiúscula	a = "python" a.capitalize() 'Python'
count()	Informa quantas vezes um caractere (ou uma sequência de caracteres) aparece na string.	b = "Linguagem Python" b.count("n") 2
startswith()	Verifica se uma string inicia com uma determinada sequência.	c = "Python" c.startswith("Py") True

# Números

- Os quatro tipos numéricos simples, utilizados em Python, são números inteiros (**int**), números longos (**long**), números decimais (**float**) e números complexos (**complex**);
- A linguagem Python também possui operadores aritméticos, lógicos, de comparação e de bit;



# Dicionários

- Dicionários são coleções de elementos onde é possível utilizar um índice de qualquer tipo imutável.
- Os dicionários implementam mapeamentos que são coleções de associações entre pares de valores:
  - O primeiro elemento é a chave;
  - O segundo elemento é o conteúdo/valor

# Dicionários

- Criação do Dicionário

- `dic = {"Nome":'Sebastiao',"Sobrenome":'Rogerio'}`
- `dic = {"Kayo":'001',"Henrique":'002',"Sebastiao":'003',"Rogerio":'004'}`

# Dicionários

- **Operações com Dicionário**

- `print(dic["Nome"])` - Imprime o conteúdo da chave Nome
- `print(dic["Sobrenome"])` - Imprime o conteúdo da chave Sobrenome
- `print(dic.keys())` - Imprime apenas as chaves
- `print(dic.values())` - Imprime apenas os conteúdos
- `print(dic.items())` - Imprime as chaves e conteúdos

# Dicionários

- **Inserindo um novo item no dicionário**
  - `dic["Idade"] = '18'`
- **Alterando o valor das chaves**
  - `dic["Nome"] = 'Rose'`

# Dicionários

- **Função GET:** retorna o valor da chave e **NONE** caso não exista
  - `print(dic.get('Kayo'))`
  - `print(dic.get('Sebastiao'))`
- **Função DEL:** Apaga determinado item do dicionário
  - `del dic["Nome"]`

# Dicionários

- **Função CLEAR:** Apaga todo o dicionário
  - `dic.clear()`
- **Função COPY:** Copia o conteúdo de um dicionário para outro
  - `dic2 = dic.copy()`

## Dicionários (ex)

```
listaprof = {"Kayo":202208,"Sebastiao":202209,"Jadson":202210}
```

```
print(listaprof["Kayo"])  
print(listaprof["Jadson"])  
print(listaprof["Sebastiao"])  
print(listaprof.keys())  
print(listaprof.values())
```

**Qual será a saída?**

## Dicionários (ex)

```
listaprof = {"Kayo":202208,"Sebastiao":202209,"Jadson":202210}
```

<pre>print(listaprof["Kayo"])</pre>	<pre>-&gt; 202208</pre>
<pre>print(listaprof["Jadson"])</pre>	<pre>-&gt; 202210</pre>
<pre>print(listaprof["Sebastiao"])</pre>	<pre>-&gt; 202209</pre>
<pre>print(listaprof.keys())</pre>	<pre>-&gt; Kayo, Sebastiao, Jadson</pre>
<pre>print(listaprof.values())</pre>	<pre>-&gt; 202208, 202209, 202210</pre>



## Exercicio

Faça um dicionário que contenha os dados de uma pessoa, são os seguintes dados: (Preencha os dados iniciais como preferir)

- Nome
- Ultimo Nome
- Idade
- Curso
- Endereço

## Exercicio

- a) Imprima o dicionário completo
- b) Imprima cada um dos 5 itens separadamente
- c) Exclua a chave CURSO do dicionário
- d) Altere o ULTIMO NOME para Lopes
- e) Imprima novamente o dicionário completo
- f) Imprima apenas o endereço
- g) Crie uma cópia do dicionário e altere Nome e Idade
- h) Imprima o segundo dicionário completo

## Exercicio (?solução?)

```
dic={"nome":"Kayo","ultimonome":"Monteiro","idade":"29","curso":"EngComputacao","endereco":"Rua Caruaru"}
```

```
print(dic.items())      #Resposta A
print(dic["nome"])      #Resposta B
print(dic["ultimonome"]) #Resposta B
print(dic["idade"])     #Resposta B
print(dic["curso"])     #Resposta B
print(dic["endereco"])  #Resposta B
```

## Exercicio (?solução?)

```
del dic["curso"]
```

#Resposta C

```
dic["ultimonome"] = "Monteiro"
```

#Resposta D

```
print(dic.items())
```

#Resposta E

```
print(dic["endereco"])
```

#Resposta F

## Exercicio (?solução?)

```
dic2 = dic.copy() #Resposta G
```

```
dic2["nome"] = "Sebastiao" #Resposta G
```

```
dic2["idade"] = "29" #Resposta G
```

```
print(dic2.items()) #Resposta H
```

# Estrutura de decisão



# O que são estruturas de decisão?

Em Python, assim como na maioria das linguagens de programação, o programa deve ser capaz de **tomar decisões** com base em valores e resultados gerados durante sua execução, ou seja, deve ser capaz de decidir se determinada instrução deve ou não ser executada de acordo com uma condição

# O que são estruturas de decisão?

Para atender a esse tipo de situação, podemos utilizar instruções especiais denominadas estruturas condicionais ou estruturas de decisão.



# Estruturas de condição com o IF

O IF é uma estrutura de condição que permite avaliar uma expressão e, de acordo com seu resultado, executar uma determinada ação.

# Estruturas de condição com o IF

- No código a seguir, temos um exemplo de uso do IF no qual verificamos se a variável idade é menor que 20.
- Em caso positivo, imprimimos uma mensagem na tela, e em caso negativo, o código seguirá normalmente, desconsiderando a linha 3.



```
idade = 18
if idade < 20:
    print('Você é jovem!')
```

# Entendendo a estrutura

- Essa estrutura é formada pela palavra reservada `if`, seguida por uma condição e por dois pontos (`:`). As linhas abaixo dela formam o bloco de instruções que serão executadas se a condição for atendida;
- Para isso, elas devem ser indentadas corretamente, respeitando a especificação do Python

# Operadores de comparação

- São essenciais para as estruturas de condição e repetição;
- Como o nome sugere, eles são usados para avaliar o valor de duas ou mais expressões/variáveis e compará-las. No exemplo anterior comparamos a variável idade com o valor 20, por meio do operador menor que (<)

# Operadores de comparação

- Na Tabela podemos ver todos os operadores de comparação da linguagem Python e seu significado/aplicação.

Símbolo	Definição
==	Igual
!=	Diferente
>	Maior que
<	Menor que
>=	Maior ou igual que
<=	Menor ou igual que


# Estrutura de condição IF-ELIF-ELSE

E quando a condição não é atendida, o que fazer?

- Quando isso é necessário, precisamos utilizar a palavra reservada **else**.
- Adicionalmente, se existir mais de uma condição alternativa que precisa ser verificada, devemos utilizar o **elif** para avaliar as expressões intermediárias antes de usar o **else**.

# Estrutura de condição IF-ELIF-ELSE

Observe o código a seguir:



```
idade = int(input('Digite sua idade: '))
if idade >= 10 and idade < 20:
    print('Você é adolescente')
elif idade >= 20 and idade < 30:
    print('Você é jovem')
elif idade >= 30 and idade <= 100:
    print('Você é adulto')
else:
    print('Valor não encontrado!')
```

Na linha 2, verificamos se o valor informado está dentro de uma faixa de valores específica.

Caso a condição seja satisfeita, o programa executará a linha 3.

Caso o resultado não seja o esperado, então o programa verificará o próximo condicional, na linha 4 e, caso ele seja verdadeiro, a linha 5 será executada

O mesmo ocorre para a verificação da linha 6. Por fim, se nenhuma das condições foi satisfeita, o programa executará o que é especificado no bloco else

## Exercicio

1. Faça um programa que peça dois números e imprima o maior deles.
2. Faça um script que peça um valor e mostre na tela se o valor é positivo ou negativo.
3. Crie um programa que verifique se uma letra digitada é "F" ou "M". Conforme a letra escrever: F - Feminino, M - Masculino, Sexo Inválido.



# Exercicio (?solução?)

```
num1=int(input('Digite o primeiro numero: '))
num2=int(input('Digite o segundo numero: '))

if num1 > num2 :
    print('O primeiro, %d, é maior' %num1)
else:
    if num1 == num2 :
        print('Os números são iguais')
    else:
        print('O segundo, %d, é maior' %num2)
```

Veja outro exemplo utilizado o operador %d

```
idade = 10
print ("Sebastião tem %d anos" %idade)
```

Obs.: O operador %d é usado como string de formatação em Python. É um espaço reservado para um número inteiro.

# Exercicio

## (?solução?)



```
num=int(input('Digite um numero: '))

if num > 0 :
    print('Positivo')
else:
    if num == 0 :
        print('Nem positivo nem
negativo, é 0')
    else:
        print('Negativo')
```

# Exercicio

## (?solução?)



```
resposta=input('M ou F: ')\n\nif resposta == 'M':\n    print('Masculina')\nelse:\n    if resposta == 'F':\n        print('Feminina')\n    else:\n        print('Você não digitou M ou F')
```

# Exercício – IF

Escreva um código em Python que leia a velocidade do vento aferida durante uma tempestade e apresente ao usuário a classificação do furacão caso a tempestade possa ser classificada como tal.

Classificação

tempestade fraca

tempestade tropical

furacão de categoria 1

furacão de categoria 2

furacão de categoria 3

furacão de categoria 4

furacão de categoria 5

Velocidade dos ventos (km/h)

abaixo de 62

62 a 118

119 a 153

154 a 177

178 a 209

210 a 249

maior que 249

# Estrutura de repetição




# O que são estruturas de repetição?

É um recurso das linguagens de programação responsável por executar um bloco de código repetidas vezes enquanto determinada condição é atendida. No Python, possuímos dois tipos de estruturas de repetição: `for` e `while`.

# Estrutura While

- O comando while faz com que um conjunto de instruções seja executado enquanto uma condição é atendida.
- Quando o resultado dessa condição passa a ser falso, a execução do loop é interrompida, como mostra o exemplo a seguir



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

# Estrutura While

- Neste código, enquanto a variável contador, inicializada com 0, for menor do que 5, as instruções das linhas 3 e 4 serão executadas.
- Observe que na linha 4 incrementamos o valor da variável contador, de forma que em algum momento seu valor igualará o número 5.



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



# Estrutura While

- Quando isso for verificado na linha 2, o laço será interrompido.
- Sem esse código, a condição de parada não será atingida, gerando o que é chamado de loop infinito.
- Evite que isso aconteça, pois leva ao congelamento e finalização da aplicação.

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

# Estrutura While-else

- Ao final do while podemos utilizar a instrução else.
- O propósito disso é executar alguma instrução ou bloco de código ao final do loop, como podemos ver no exemplo a seguir:

```
contador = 0
while (contador < 5):
    print(contador)
    contador = contador + 1
else:
    print("O loop while foi encerrado com sucesso!")
```

# Estrutura While-else

- Assim como acontece com for/else, declarando o else ao final do while é possível executar um código ao final do loop.
- Neste caso será impressa a mensagem: "O loop while foi encerrado com sucesso!".



```
contador = 0
while (contador < 5):
    print(contador)
    contador = contador + 1
else:
    print("O loop while foi encerrado com sucesso!")
```

# Estrutura While-else

- No loop while, a expressão é testada enquanto for verdadeira.
- A partir do momento que ela se torna falsa, o código da cláusula **else** será executado, se estiver presente.



```
x = 0
while x < 10:
    print(x)
    x += 1
else:
    print("fim while")
```

# Estrutura While-else

- Se dentro da repetição for executado um break, o loop será encerrado sem executar o conjunto da cláusula else.

```
x = 0
while x < 10:
    print(x)
    x += 1
    if x == 6:
        print("x é igual a 6")
        break
else:
    print("fim while")
```

# Referências

- <https://www.devmedia.com.br/python-estrutura-de-repeticao-while/38546#:~:text=A%20estrutura%20de%20repeti%C3%A7%C3%A3o%20%C3%A9,documenta%C3%A7%C3%A3o%20abordaremos%20o%20comando%20while.>
- <https://www.devmedia.com.br/estruturas-de-condicao-em-python/37158>

# To be...



CREDITS: This presentation template was created by Slidesgo, including icons by Flaticon, and infographics & images by Freepik.