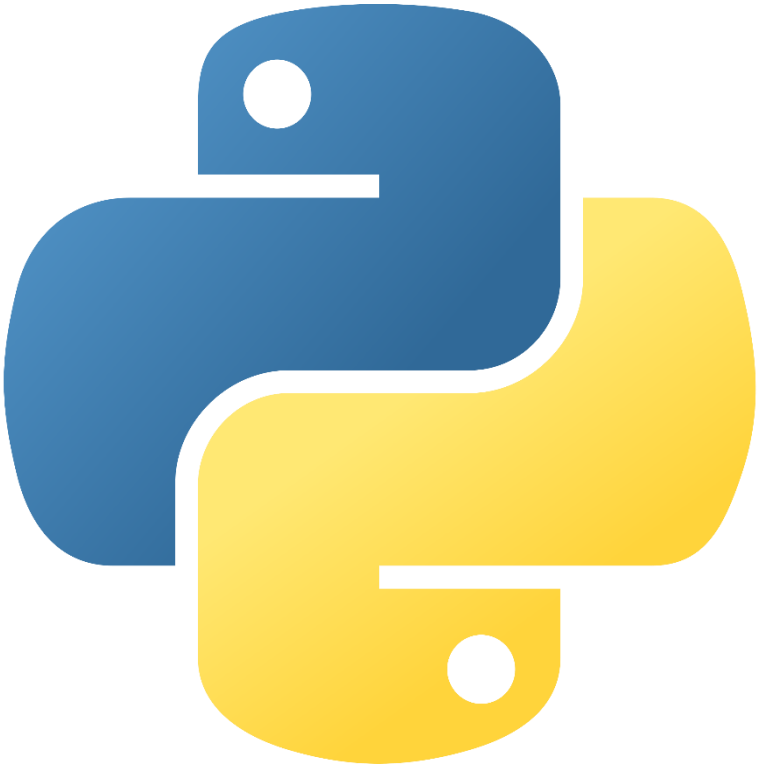

python™

Jefferson Martins de Moura

Abril de 2024

Sumário

## OBJETIVO

Meu objetivo é fornecer uma abordagem prática para o aprendizado dos fundamentos da programação lógica utilizando a linguagem Python, com foco na manipulação eficiente de dados. Minha proposta inclui uma variedade de exemplos, exercícios e projetos que abrangem desde conceitos básicos até desafios mais avançados, visando atender a todos os níveis de interesse e habilidade.

Valorizo o seu interesse em se aprofundar em projetos mais específicos e complexos. Portanto, estou comprometido em compartilhar irrestritamente informações e recursos relevantes, de acordo com as suas necessidades e interesses individuais. Entretanto o seu interesse é a energia que preciso para manter as engrenagens girando.

Estou empenhado em proporcionar um ambiente de aprendizado colaborativo e enriquecedor, onde você tem a oportunidade de explorar, aprender e crescer em sua jornada de programação com Python.

Para mim, a vida é um constante processo de aprendizado e crescimento. Acredito firmemente no lema 'Aprender para viver e viver para ensinar!' Essas palavras encapsulam minha abordagem para enfrentar os desafios da vida. Busco constantemente adquirir novos conhecimentos e experiências, não apenas para meu próprio benefício, mas também para compartilhar esse conhecimento com os outros. Acredito que ensinar e aprender estão intrinsecamente ligados, e é através desse ciclo de aprendizado e ensino que encontro significado e propósito em minha jornada.

## INFORMAÇÕES GERAIS

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, Python é conhecida por sua sintaxe simples e legibilidade, o que a torna ideal tanto para iniciantes quanto para desenvolvedores experientes. Amplamente utilizado em uma variedade de domínios, incluindo desenvolvimento web, ciência de dados, automação de tarefas, desenvolvimento de jogos e muito mais. Sua popularidade e versatilidade fazem dela uma escolha popular entre programadores de todos os níveis de habilidade.

Uma IDE (Integrated Development Environment) ou Ambiente de Desenvolvimento Integrado, é um software que combina várias ferramentas e funcionalidades em uma única interface para auxiliar no desenvolvimento de software.

IDE para o treinamento:

Online:

https://www.online-python.com/

https://www.onlinegdb.com/online_python_compiler

https://www.programiz.com/python-programming/online-compiler/

Offline:

PyCharm: Desenvolvido pela JetBrains, o PyCharm é uma IDE poderosa e amplamente utilizada para desenvolvimento Python. Ele oferece uma gama completa de recursos, incluindo realce de sintaxe avançado, depurador integrado, suporte a testes unitários, gerenciamento de projetos e integração com controle de versão. O PyCharm está disponível em uma versão gratuita (Community) e uma versão paga (Professional), com funcionalidades adicionais para desenvolvimento profissional.

Visual Studio Code (VSCode): Desenvolvido pela Microsoft, o Visual Studio Code é uma IDE leve e altamente personalizável que ganhou popularidade entre os desenvolvedores Python. Ele oferece suporte a extensões, o que permite adicionar funcionalidades específicas para Python, como realce de sintaxe, depurador, controle de versão e suporte a frameworks populares como Django e Flask.

Spyder: Spyder é uma IDE dedicada para computação científica e análise de dados em Python. Ele oferece recursos específicos para desenvolvimento científico, como integração com bibliotecas populares de análise de dados (NumPy, Pandas, Matplotlib), visualização de variáveis, edição de notebooks Jupyter e suporte a análise de dados em tempo real.

Essas são apenas algumas das IDEs mais populares para programar em Python. Cada uma tem suas próprias vantagens e recursos únicos, portanto, é uma boa ideia experimentar algumas delas para descobrir qual se adapta melhor às suas necessidades e preferências de desenvolvimento.

Para alguns recursos os Online vai ser suficiente, mas para melhor experiencia ter um dos interpretadores Offline seria ideal para o uso de bibliotecas para usar o máximo dos recursos.

## SUGESTÃO DE VÍDEOS:

**Observação**: Os vídeos destacados foram escolhidos pela abordagem e repetição para melhor entendimento, sem nenhuma parceria ou relacionamento financeiro.

Canal Youtube - **Curso em vídeo**:

https://www.youtube.com/watch?v=S9uPNppGsGo&list=PLvE-ZAFRgX8hnECDn1v9HNTI71veL3oW0

Canal Youtube - **Hashtag Programação**:

https://www.youtube.com/watch?v=4p7axLXXBGU

https://www.youtube.com/watch?v=BW9Va5syNC0&list=PLpdAy0tYrnKyCZsE-ifaLV1xnkXBE9n7T

Canal Youtube - **Refatorando**:
https://www.youtube.com/watch?v=Jk4kuuwLThA&list=PLj7gJIFoP7jdirAFg-fHe9HKOnGLGXSHZ

## STRINGS

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().

**Observação:** Em alguns casos o uso das duas será necessário, mas somente uma delas ficara externamente e outra sempre interno. *"Exemplo do 'uso' das 'aspas'."*.

## VARIÁVEIS

Variáveis são pequenos espaços de memória, utilizados para armazenar e manipular dados. Em Python, os tipos de dados básicos são: tipo inteiro (armazena números inteiros), tipo float (armazena números em formato decimal), e tipo string (armazena um conjunto de caracteres). 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. Quando se faz uma atribuição de valor, automaticamente a variável se torna do tipo do valor armazenado, como apresentado nos exemplos a seguir:

.

Exemplos:

A = 10

Print(A)

Resultado = 10

**Observação:** A variável “**A**” se torna uma variável do tipo inteiro.

B = 1.2

Print(B)

Resultado = 1.2

**Observação:** A variável “**B**” se torna uma variável do tipo float.

C = “COSR-SE”

Print(C)

Resultado = “COSR-SE”

**Observação:** A variável “C” se torna uma variável do tipo string.

**Spoiler:**

O comando “print” (imprimir), exibe o valor da variável nela apresentada, podendo apresentar mais de uma variável ao mesmo tempo separando-as com “,” (virgula).

O comando “input” (entrada), recebe uma informação para ser vinculada a uma variável, mas isso quando estamos trabalhando apenas em terminal de comandos.

Exercício:

1) Crie uma variável chamada “N1” e atribua o valor 10 e crie uma variável chamada “entrada” e atribua o comando abaixo:
   N1 = 10
   entrada = float(input(‘Digite um número para ser multiplicado por 10: ‘))
   resultado = entrada * N1
   print(resultado)

2) Se para calcular o IMC é ‘peso’ dividido pela ‘altura²’ como poderíamos fazer?

   Obs.: as variáveis numéricas devem ser ‘float’

Tabela 1 - Exemplos de nomes válidos e inválidos

| Nomes | 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. |

**Dica**: evite colocar acentos e preferencialmente letras minúsculas, para não confundir com comandos.

## MANIPULAÇÃO DE STRINGS

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

Tabela 2 - Manipulação de strings

| Método | Descrição | Exemplo |
|---|---|---|
| len() | Retorna o tamanho da string. | teste = “Apostila de Python”<br>len(teste)<br>18 |
| capitalize() | Retorna a string com a primeira letra maiúscula | a = "python"<br>a.capitalize()<br>'Python' |
| count() | Informa quantas vezes um caractere (ou uma sequência de caracteres) aparece na string. | b = "Linguagem Python"<br>b.count("n")<br>2 |
| startswith() | Verifica se uma string inicia com uma determinada sequência. | c = "Python"<br>c.startswith("Py")<br>True |
| endswith() | Verifica se uma string termina com uma determinada sequência. | d = "Python"<br>d.endswith("Py")<br>False |
| isalnum() | Verifica se a string possui algum conteúdo alfanumérico (letra ou número). | e = "!@#$%"<br>e.isalnum()<br>False |
| isalpha() | Verifica se a string possui apenas conteúdo alfabético. | f = "Python"<br>f.isalpha()<br>True |

| | | |
|---|---|---|
| islower() | Verifica se todas as letras de uma string são minúsculas. | g = "pytHon"<br>g.islower()<br>False |
| isupper() | Verifica se todas as letras de uma string são maiúsculas. | h = "# PYTHON 12"<br>h.isupper()<br>True |
| lower() | Retorna uma cópia da string trocando todas as letras para minúsculo. | i = "#PYTHON 3"<br>i.lower()<br>'#python 3' |
| upper() | Retorna uma cópia da string trocando todas as letras para maiúsculo. | j = "Python"<br>j.upper()<br>'PYTHON' |
| swapcase() | Inverte o conteúdo da string (Minúsculo / Maiúsculo). | k = "Python"<br>k.swapcase()<br>'pYTHON' |
| title() | Converte para maiúsculo todas as primeiras letras de cada palavra da string. | l = "apostila de python"<br>l.title()<br>'Apostila De Python' |
| split() | Transforma a string em uma lista, utilizando os espaços como referência. | m = "cana de açúcar"<br>m.split()<br>['cana', 'de', 'açúcar'] |
| replace(S1, S2) | Substitui na string o trecho S1 pelo trecho S2. | n = "Apostila teste"<br>n.replace("teste", "Python")<br>'Apostila Python' |
| find() | Retorna o índice da primeira ocorrência de um determinado caractere na string. Se o caractere não estiver na string retorna -1. | o = "Python"<br>o.find("h")<br>3 |
| ljust() | Ajusta a string para um tamanho mínimo, acrescentando espaços à direita se necessário. | p = " Python"<br>p.ljust(15)<br>' Python ' |
| rjust() | Ajusta a string para um tamanho mínimo, acrescentando espaços à esquerda se necessário. | q = "Python"<br>q.rjust(15)<br>' Python' |
| center() | Ajusta a string para um tamanho mínimo, acrescentando espaços à esquerda e à direita, se necessário. | r = "Python"<br>r.center(10)<br>' Python ' |
| lstrip() | Remove todos os espaços em branco do lado esquerdo da string. | s = " Python "<br>s.lstrip()<br>'Python ' |
| rstrip() | Remove todos os espaços em branco do lado direito da string. | t = " Python "<br>t.rstrip()<br>' Python' |
| strip() | Remove todos os espaços em branco da string. | u = " Python "<br>u.strip()<br>'Python' |

## FATIAMENTO DE STRINGS

O fatiamento é uma ferramenta usada para extrair apenas uma parte dos elementos de uma string.

Nome_String [Limite_Inferior : Limite_Superior]

Retorna uma string com os elementos das posições do limite inferior até o limite superior -1.

Exemplo:
s = "Python"
s[1:4]  *seleciona os elementos das posições 1,2,3
Resultado: 'yth'
s[2:]  *seleciona os elementos a partir da posição 2
Resultado: 'thon'
s[:4]  *seleciona os elementos até a posição 3
Resultado: 'Pyth'

Exercício:

3) Descubra o resultado das variáveis abaixo utilizando [2:3]

   Var1 = ‘cosr-se’

   Var2 = ‘sala de controle’

   Var3 = 34587123

4) Considere a string A = "Um elefante incomoda muita gente". Que fatia corresponde a "elefante incomoda"?

## 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.

Tabela 3 - Operadores Aritméticos

| Operador | Descrição | Exemplo |
|---|---|---|
| + | Soma | 5 + 5 = 10 |
| - | Subtração | 7 – 2 = 5 |
| * | Multiplicação | 2 * 2 = 4 |
| / | Divisão | 4 / 2 = 2 |
| % | Resto da divisão | 10 % 3 = 1 |
| ** | Potência | 4 ** 2 = 16 |

Tabela 4 - Operadores de Comparação

| Operador | Descrição | Exemplo |
|---|---|---|
| < | Menor que | a < 10 |
| <= | Menor ou igual | b <= 5 |
| > | Maior que | c > 2 |
| >= | Maior ou igual | d >= 8 |
| == | Igual | e == 5 |
| != | Diferente | f != 12 |

Tabela 5 - Operadores Lógicos

| Operador | Descrição | Exemplo |
|---|---|---|
| not | NÃO | not a |
| and | E | (a <=10) and (c = 5) |
| or | OU | (a <=10) or (c = 5) |

**Dica**: Geralmente ao adicionar o valor da variável o python tenta interpretar da melhor forma possível assumindo que o valor venha ser “int, float, long, string e etc.”, mas caso precise de uma característica única sem chance de erros, adicione por você mesmo. Se o valor é “10” então a variável é int, mas podemos colocar int(10) para não correr riscos dependendo do seu objetivo.

Exercício:

5) Escreva um programa que receba 2 valores do tipo inteiro ‘x’ e ‘y’, e calcule o valor de z:
$z = \frac{(x^2+y^2)}{(x-y)^2}$

6) Escreva um programa que receba o salário de um funcionário (float), e retorne o resultado do novo salário com reajuste de 35%.

## LISTAS

Lista é um conjunto sequencial de valores, onde cada valor é identificado através de um índice. O primeiro valor tem índice ‘0’. Uma lista em Python é declarada da seguinte forma:

Nome_Lista= [ valor1, valor2, ..., valorN]

Uma lista pode ter valores de qualquer tipo, incluindo outras listas.

Exemplo:

```
L = [3 ,'abacate' ,9.7 , [5,6,3] ,"Python" , (3, 'j')]
print(L[2])
9.7
print(L[3])
[5,6,3]
print(L[3][1])
6
```

Para alterar um elemento da lista, basta fazer uma atribuição de valor através do índice. O valor existente será substituído pelo novo valor.

Exemplo:

```
L[3]= 'morango'
print(L)
L = [3 , 'abacate' , 9.7 , 'morango', "Python" , (3 , 'j')]
```

A tentativa de acesso a um índice inexistente resultará em erro.

```
L[7]= 'banana'
```

### Funções para manipulação de listas

A lista é uma estrutura mutável, ou seja, ela pode ser modificada. Na tabela a seguir estão algumas funções utilizadas para manipular listas.

Tabela 6 - Operações com listas

| Função | Descrição | Exemplo |
|---|---|---|
| len | retorna o tamanho da lista. | L = [1, 2, 3, 4]<br>len(L) -> 4 |
| min | retorna o menor valor da lista. | L = [10, 40, 30, 20]<br>min(L) -> 10 |

| | | |
|---|---|---|
| max | retorna o maior valor da lista. | L = [10, 40, 30, 20]<br>max(L) -> 40 |
| sum | retorna soma dos elementos da lista. | L = [10, 20, 30]<br>sum(L) -> 60 |
| append | adiciona um novo valor na no final da lista. | L = [1, 2, 3]<br>L.append(100)<br>L -> [1, 2, 3, 100] |
| extend | insere uma lista no final de outra lista. | L = [0, 1, 2]<br>L.extend([3, 4, 5])<br>L -> [0, 1, 2, 3, 4, 5] |
| del | remove um elemento da lista, dado seu índice. | L = [1,2,3,4]<br>del L[1]<br>L-> [1, 3, 4] |
| in | verifica se um valor pertence à lista. | L = [1, 2 , 3, 4]<br>3 in L -> True |
| sort() | ordena em ordem crescente | L = [3, 5, 2, 4, 1, 0]<br>L.sort()<br>L -> [0, 1, 2, 3, 4, 5] |
| reverse() | inverte os elementos de uma lista. | L = [0, 1, 2, 3, 4, 5]<br>L.reverse()<br>L -> [5, 4, 3, 2, 1, 0] |

O fatiamento de listas é semelhante ao fatiamento de strings.

**Operações com listas**

Concatenação ( + )

```
a = [0,1,2]
b = [3,4,5]
c = a + b
print(c)
[0, 1, 2, 3, 4, 5]
```

Repetição ( * )

```
L = [1,2]
R= L * 4
print(R)
[1, 2, 1, 2, 1, 2, 1, 2]
```

## Fatiamento de listas

O fatiamento de listas é semelhante ao fatiamento de strings.

## Exemplo:

```
L = [3,'abacate',9.7,[5,6,3],"Python",(3,'j')]
L[1:4]  *seleciona os elementos das posições 1,2,3

['abacate', 9.7, [5, 6, 3]]
```

L[2:]  *seleciona os elementos a partir da posição 2

[9.7, [5, 6, 3], 'Python', (3, 'j')]
L[:4]   *seleciona os elementos até a posição 3

[3, 'abacate', 9.7, [5, 6, 3]]

## Criação de listas com ‘range()’

A função ‘range()’ define um intervalo de valores inteiros. Associada a list(), cria uma lista com os valores do intervalo.

A função range() pode ter de 1 a 3 parâmetros:

* range(n) * gera um intervalo de 0 a n-1
* range(i , n) * gera um intervalo de i a n-1
* range(i , n, p) * gera um intervalo de i a n-1 com intervalo p entre os números

## Exemplos:

```
L1 = list(range(5))
print(L1)
[0, 1, 2, 3, 4]

L2 = list(range(3,8))
print(L2)
[3, 4, 5, 6, 7]

L3 = list(range(2,11,3))
print(L3)
[2, 5, 8]
```

## Exercícios:

7) Dada a lista L= [5, 7, 2, 9, 4,1, 3], escreva um programa que imprima as seguintes informações:
   a) tamanho da lista.
   b) maior valor da lista.
   c) menor valor da lista.
   d) soma de todos os elementos da lista.
   e) lista em ordem crescente.
   f) lista em ordem decrescente.

## TUPLAS

Tupla, assim como a Lista, é um conjunto sequencial de valores, onde cada valor é identificado através de um índice. A principal diferença entre elas é que as tuplas são imutáveis, ou seja, seus elementos não podem ser alterados.

Dentre as utilidades das tuplas, destacam-se as operações de empacotamento e desempacotamento de valores.

Uma tupla em Python é declarada da seguinte forma:
Nome_tupla = (valor1, valor2, ..., valorN)

Exemplo:

```
T = (1,2,3,4,5)
print(T)
(1, 2, 3, 4, 5)

print(T[3])
4
```

Exercício:

8) Dado a variável abaixo T qual o resultado esperado se fizer o comando print(T[3]=8)?

```
T = (1,2,3,4,5)
T[3] == 8
Print(?)
```

Uma ferramenta muito utilizada em tuplas é o desempacotamento, que permite atribuir os elementos armazenados em uma tupla a diversas variáveis.

Exemplo:

```
T = (10,20,30,40,50)

a,b,c,d,e = T

print("a=",a,"b=",b)

a= 10 b= 20
```

```
print("d+e=",d+e)

d+e= 90
```

## DICIONÁRIOS

Dicionário é um conjunto de valores, onde cada valor é associado a uma chave de acesso.

Um dicionário em Python é declarado da seguinte forma:

```
Nome_dicionario = {  chave1: valor1,
chave2: valor2,
chave3: valor3,
......
chaveN: valorN}
```

Exemplo:

```
D = {"arroz": 17.30, "feijão": 12.50, "carne": 23.90, "alface": 3.40}
print(D)
{'arroz': 17.3, 'carne': 23.9, 'alface': 3.4, 'feijão': 12.5}

print(D["carne"])
23.914
```

Exercício:

9) No dicionário abaixo qual o resultado se fizer o ‘print(dicionario[“carlos”]’?

   Dicionario = {'ana': 456, 'bruno': 789, 'cátia': 234, 'daniel': 567, 'eduarda': 890, 'fernando': 123, 'gabriela': 678, 'hugo': 901, 'inês': 345, 'joão': 678, 'karen': 901, 'luís': 234, 'mariana': 567, 'nuno': 890, 'olívia': 123, 'pedro': 456, 'rita': 789, 'sérgio': 234, 'tânia': 567, 'ursula': 890, 'vasco': 123, 'wanda': 456, 'xavier': 789, 'yara': 901, 'zélia': 234, 'andreia': 567, 'bernardo': 890, 'diana': 456, 'eva': 789}

Os valores do dicionário não possuem ordem, por isso a ordem de impressão dos valores não é sempre a mesma.

## OPERAÇÕES EM DICIONÁRIOS

Na tabela 7 são apresentados alguns comandos para a manipulação de dicionários.

Tabela 7 – Comandos em dicionários

| Comando | Descrição | Exemplo |
|---|---|---|
| del | Exclui um item informando a chave. | Del D[“feijão”]<br>print(D)<br>{‘alface’:3.4<br>‘tomate’:8.8,’arroz’:17.3,’carne’:25.0} |
| in | Verificar se uma chave existe no dicionário. | “batata” in D<br>False<br>“alface” in D<br>True |
| keys() | Obtém as chaves de um dicionário. | D.keys()<br>dict_keys([‘alface’, ‘tomate,’carne’, ‘arroz’]) |
| values() | Obtém os valores de um dicionário. | D.values()<br>dict_values([3.4, 8.8, 25.0, 17.3]) |

Os dicionários podem ter valores de diferentes tipos.

Exemplo:

```
Dx ={2:'carro', 3:[4,5,6], 7:('a','b'),4:173.8}
print(Dx[7])
```

(‘a’, ‘b’)

Exercícios:

10) Dada a tabela a seguir, crie um dicionário que a represente:

| Lanchonete | |
|---|---|
| Produtos | Preços R$ |
| Salgado | R$ 4.50 |
| Lanche | R$ 6.50 |
| Suco | R$ 3.00 |
| Refrigerante | R$ 3.50 |
| Doce | R$ 1.00 |

## Exercícios de revisão:

### Strings:

1. **Concatenação de Strings:**

   Solicite ao usuário que insira seu primeiro nome e sobrenome separadamente e depois concatene-os para formar o nome completo.

2. **Contagem de Caracteres:**

   Peça ao usuário que insira uma palavra e conte quantos caracteres a palavra possui.

3. **Maiúsculas e Minúsculas:**

   Peça ao usuário para inserir uma frase e converta-a completamente para letras minúsculas.

4. **Substituição de Caracteres:**

   Solicite ao usuário que insira uma frase e substitua todas as ocorrências de uma determinada letra por outra letra especificada.

5. **Divisão de Palavras:**

   Peça ao usuário que insira uma frase e divida-a em uma lista de palavras.

### Variáveis:

1. **Atribuição Simples:**

   Atribua um número inteiro a uma variável e imprima seu valor.

2. **Atribuição Múltipla:**

   Atribua valores a múltiplas variáveis em uma única linha e depois imprima-as.

3. **Retribuição de Variáveis:**

   Atribua um valor a uma variável, depois mude esse valor e imprima a variável novamente.

4. **Variáveis de Texto:**

   Atribua uma string a uma variável e imprima-a.

5. **Variáveis Numéricas:**

   Atribua valores numéricos (inteiros ou de ponto flutuante) a variáveis e realize operações aritméticas simples com elas.

## Manipulação de String:

1. **Capitalização:**

   Converta uma string para todos os caracteres em maiúsculas.

2. **Substituição:**

   Substitua todas as ocorrências de uma palavra em uma frase por outra palavra.

3. **Contagem de Substrings:**

   Conte quantas vezes uma determinada substring ocorre em uma string.

4. **Separação de String:**

   Separe uma string em substrings com base em um caractere de delimitação.

5. **Remoção de Espaços:**

   Remova todos os espaços em branco de uma string.

## Fatiamento de String:

1. **Fatiamento Simples:**

   Dada uma string, imprima apenas os primeiros três caracteres.

2. **Fatiamento com Intervalo:**

   Imprima uma substring de uma string dada usando um intervalo.

3. **Fatiamento Reverso:**

Imprima uma substring de uma string de trás para frente.

4. **Fatiamento Alternado:**

Imprima uma substring de uma string pulando caracteres em intervalos específicos.

5. **Fatiamento com Índices Negativos:**

Imprima uma substring de uma string usando índices negativos.

## Números (Operações aritméticas, comparação, lógico):

1. **Operações Aritméticas Simples:**

Realize operações de adição, subtração, multiplicação e divisão entre dois números.

2. **Comparação de Números:**

Compare dois números e imprima se são iguais, diferentes, maiores ou menores.

3. **Operações Lógicas:**

Realize operações lógicas (E, OU, NÃO) entre duas variáveis booleanas.

4. **Arredondamento de Números:**

Arredonde um número de ponto flutuante para um número inteiro.

5. **Conversão de Tipos:**

Converta um número inteiro para ponto flutuante e vice-versa, depois imprima-os.

## Listas:

1. **Criação de Lista:**

Crie uma lista com alguns números inteiros e imprima-a.

2. **Acesso a Elementos:**

   Acesse e imprima o terceiro elemento de uma lista.

3. **Adição de Elementos:**

   Adicione um novo elemento ao final de uma lista e imprima a lista atualizada.

4. **Remoção de Elementos:**

   Remova o segundo elemento de uma lista e imprima a lista resultante.

5. **Tamanho da Lista:**

   Imprima o número de elementos presentes em uma lista.

## Operações com Listas:

1. **Concatenação de Listas:**

   Concatene duas listas e imprima a lista resultante.

2. **Ordenação de Lista:**

   Ordene os elementos de uma lista em ordem crescente e imprima-a.

3. **Reversão de Lista:**

   Inverta os elementos de uma lista e imprima-a.

4. **Contagem de Elementos:**

   Conte quantas vezes um determinado elemento ocorre em uma lista.

5. **Cópia de Lista:**

   Copie uma lista para outra variável e imprima ambas para verificar se são iguais.

## Fatiamento de Lista:

1. **Fatiamento Simples:**

Dada uma lista, imprima apenas os primeiros três elementos.

2. **Fatiamento com Intervalo:**

Imprima uma sublista de uma lista dada usando um intervalo.

3. **Fatiamento Reverso:**

Imprima uma sublista de uma lista de trás para frente.

4. **Fatiamento Alternado:**

Imprima uma sublista de uma lista pulando elementos em intervalos específicos.

5. **Fatiamento com Índices Negativos:**

Imprima uma sublista de uma lista usando índices negativos.

## Tuplas:

1. **Criação de Tupla:**

Crie uma tupla com alguns valores e imprima-a.

2. **Acesso a Elementos:**

Acesse e imprima o terceiro elemento de uma tupla.

3. **Concatenação de Tuplas:**

Concatene duas tuplas e imprima a tupla resultante.

4. **Comprimento da Tupla:**

Imprima o número de elementos presentes em uma tupla.

5. **Imutabilidade da Tupla:**

Tente alterar um elemento de uma tupla e observe o erro gerado.

## Dicionários:

1. **Criação de Dicionário:**

   Crie um dicionário com alguns pares chave-valor e imprima-o.

2. **Acesso a Valores:**

   Acesse e imprima o valor associado a uma chave específica em um dicionário.

3. **Adição de Novos Pares:**

   Adicione um novo par chave-valor a um dicionário

# ESTRUTURAS DE DECISÃO

As estruturas de decisão permitem alterar o curso do fluxo de execução de um programa, de acordo com o valor (Verdadeiro/Falso)de um teste lógico.

Em Python temos as seguintes estruturas de decisão:

if (se)

if..else (se..senão)

if..elif..else (se..senão se..senão)

## Estrutura ‘if’

O comando ‘if’ é utilizado quando precisamos decidir se um trecho do programa deve ou não ser executado. Ele é associado a uma condição, e o trecho de código será executado se o valor da condição for verdadeiro.

Sintaxe:

```
if <condição>:
        <Bloco de comandos >
```

### Exemplo:

```
Valor = int(input(“Qual sua idade?”))
If valor < 18:
        print(“Voçê ainda não pode dirigir!”)
```

## Estrutura if..else

Nesta estrutura, um trecho de código será executado se a condição for verdadeira e outro sea condição for falsa.

Sintaxe:

```
if<condição>:
        <Bloco de comandos para condição verdadeira>
else:
        <Bloco de comandos para condição falsa>
```

Exemplo:

```
Valor = int(input(“Qual sua idade?”))


If valor < 18:
        print(“Voçê ainda não pode dirigir!”)
else:
        print(“Voçê pode dirigir!”)
```

## Comando if..elif..else

Se houver diversas condições, cada uma associada a um trecho de código, utiliza-se o elif.

Sintaxe:

```
if<condição 1>:
        <Bloco de comandos 1>
elif<condição 2>:
        <Bloco de comandos 2>
elif<condição 3>:
        <Bloco de comandos 3>
else:
        <Bloco de comandos default>17
```

Somente o bloco de comandos associado à primeira condição verdadeira encontrada será executado. Se nenhuma das condições tiver valor verdadeiro, executa o bloco de comandos default.

Exemplo:

```
Valor = int(input(“Digite o ano atual:”))
If valor == 2024:
        print(“Esta certo: 2024”)
elIf valor == 24:
        print(“Esta certo: 2024, mas poderia ser com 4 dígitos”)
else:
        print(“Está errado!”)
```

## Exercícios: estruturas de decisão

11) Faça um programa que leia 2 notas de um aluno, calcule a média e imprima aprovado ou reprovado (para ser aprovado a média deve ser no mínimo 6)

12) Refaça o exercício 11, identificando o conceito aprovado(média superior a 6), repetência (média entre 4 e 6) ou reprovado (média inferior a 4).

# ESTRUTURAS DE REPETIÇÃO

A Estrutura de repetição é utilizada para executar uma mesma sequência de comandos várias vezes. A repetição está associada a uma condição, que indica se deve continuar ou não a repetição, ou a uma sequência de valores, que determina quantas vezes a sequência deve ser repetida. As estruturas de repetição são conhecidas também como laços (loops).

### Laço while

No laço "**while**", o trecho de código da repetição está associado a uma condição. Enquanto a condição tiver valor verdadeiro, o trecho é executado. Quando a condição passa a ter valor falso, a repetição termina.

Sintaxe:

```
while <condição>:
        <Bloco de comandos>
```

### Exemplo:

```
senha = "54321"
leitura =" "
while(leitura != senha):
        leitura = input("Digite a senha: ")
        if leitura == senha :
                print('Acesso liberado ')
        else:
                print('Senha incorreta. Tente novamente')
```

```
Digite a senha: abcde
Senha incorreta. Tente novamente
Digite a senha:12345
Senha incorreta. Tente novamente
Digite a senha:54321
Acesso liberado
```

Exemplo: Encontrar a soma de 5 valores.

```
contador = 0
somador = 0
while contador < 5:
        contador= contador + 1
        valor = float(input('Digite o '+str(contador)+'o valor: '))
        somador = somador + valor
        print('Soma = ', somador)
```

## Laço for

O laço "**for**" é a estrutura de repetição mais utilizada em Python. Pode ser utilizado com uma sequência numérica (gerada com o comando range) ou associado a uma lista. O trecho de código da repetição é executado para cada valor da sequência numérica ou da lista.

Sintaxe:

```
for<variável> in range(início, limite, passo):
        <Bloco de comandos >
```

Ou

```
for<variável> in<lista>:
        <Bloco de comandos >
```

### Exemplos:

1. Encontrar a soma S = 1+4+7+10+13+16+19

```
S=0
for x in range(1,20,3):
        S = S+x
        print('Soma = ',S)
```

2. As notas de um aluno estão armazenadas em uma lista. Calcular a média dessas notas.

```
Lista_notas= [3.4,6.6,8,9,10,9.5,8.8,4.3]
soma=0
for nota in Lista_notas:
        soma = soma+nota
        média = soma/len(Lista_notas)
print('Média= ',média)
```

### Exercícios: estrutura de repetição

13) Escreva um programa para encontrar a soma S = 3 + 6 + 9 +…. + 333.
14) Escreva um programa que leia 10 notas e informe a média dos alunos.
15) Escreva um programa que leia um número de 1 a 10, e mostre a tabuada desse número.

## BIBLIOTECAS

As bibliotecas armazenam funções pré-definidas, que podem ser utilizados em qualquer momento do programa. Em Python, muitas bibliotecas são instaladas por padrão junto com o programa. Para usar uma biblioteca, deve-se utilizar o comando import:

### Exemplo:

importar a biblioteca de funções matemáticas:
```
import math
print(math.factorial(6))
```

Pode-se importar uma função específica da biblioteca:
```
from math import factorial
print(factorial(6))
```

A tabela a seguir, mostra algumas das bibliotecas padrão de Python.

### Tabela 8 - Algumas bibliotecas padrão do Python:

| Bibliotecas | Função |
|---|---|
| math | Funções matemáticas |
| tkinter | Interface Gráfica padrão |
| smtplib | e-mail |
| time | Funções de tempo |

Além das bibliotecas padrão, existem também outras bibliotecas externas de alto nível disponíveis para Python. A tabela a seguir mostra algumas dessas bibliotecas.

Tabela 9 - Algumas bibliotecas externas para Python

| Bibliotecas | Função |
|---|---|
| urllib | Leitor de RSS para uso na internet |
| numpy | Funções matemáticas mais avançadas |
| PIL/Pillow | Manipulação de imagens |

## FUNÇÕES

Funções são pequenos trechos de código reutilizáveis. Elas permitem dar um nome a um bloco de comandos e executar esse bloco, a partir de qualquer lugar do programa.

### Como definir uma função

Funções são definidas usando a palavra-chave "**def"** (Ela é uma abreviação de "define"), conforme sintaxe a seguir:

```
def<nome_função> (<definição dos parâmetros>):
        <Bloco de comandos da função>
```

Obs.: A definição dos parâmetros é opcional.

### Exemplo:

```
Função simples
def hello():
        print("Olá Mundo!!!")
```

Para usar a função, basta chamá-la pelo nome:

```
hello()
```

Olá,  Mundo!!!

## Parâmetros e argumentos

Parâmetros são as variáveis que podem ser incluídas nos parênteses das funções. Quando a função é chamada são passados valores para essas variáveis. Esses valores são chamados argumentos. O corpo da função pode utilizar essas variáveis, cujos valores podem modificar o comportamento da função.

## Exemplo:

```
Função para imprimir o maior entre 2 valores
def maior(x,y):
        if x>y:
                print(x)
        else:
                print(y)
maior(4,7)
```

## Escopo das variáveis

Toda variável utilizada dentro de uma função tem escopo local, isto é, ela não será acessível por outras funções ou pelo programa principal. Se houver variável com o mesmo nome fora da função, será uma outra variável, completamente independentes entre si.

## Exemplo:

```
def soma(x,y):
        total = x+y
        print("Total soma = ",total)

#programa principal
total = 10
soma(3,5)
print("Total principal = ",total)
```

*Resultado da execução:
Total soma = 8
Total principal = 10

Para uma variável ser compartilhada entre diversas funções e o programa principal, ela deve ser definida como variável global. Para isto, utiliza-se a instrução global para declarara variável em todas as funções para as quais ela deva estar acessível. Isso vale para o programa principal.

Exemplo:

```
def soma(x,y):
        global total
        total = x+y
        print("Total soma = ",total)

#programa principal
global total
total = 10
soma(3,5)
print("Total principal = ",total)
```

*Resultado da execução:
Total soma = 8
Total principal = 8

## Retorno de valores

O comando “**return**” é usado para retornar um valor de uma função e encerrá-la. Caso não seja declarado um valor de retorno, a função retorna o valor ‘None’ (que significa nada, sem valor).

Exemplo:

```
def soma(x,y):
        total = x+y
        return total
        print(“feito”)

#programa principal
s=soma(3,5)
print("soma = ",s)
```

*Resultado da execução:
soma = 8

**Observações:**
a) O valor da variável total, calculado na função soma, retornou da função e foi atribuído à variável s.

b) O comando após o return foi ignorado.

### Valor padrão

É possível definir um valor padrão para os parâmetros da função. Neste caso, quando o valor é omitido na chamada da função, a variável assume o valor padrão.

### Exemplo:

```
def calcula_juros(valor, taxa=10):
        juros = valor*taxa/100
        return juros

calcula_juros(500)
50.0
```

### Exercícios: funções

16) Crie uma função para desenhar uma linha, usando o caractere'_'. O tamanho da linha deve ser definido na chamada da função.
17) Crie uma função que receba como parâmetro uma lista, com valores de qualquer tipo. A função deve imprimir todos os elementos da lista numerando-os.
18) Crie uma função que receba como parâmetro uma lista com valores numéricos e retorne a média desses valores.

## MANIPULANDO DADOS

A manipulação de dados começa na importação das informações. Caso esteja usando IDE Online aqui esta um conjunto de dados:

Copia as informações abaixo desta linha até “#selecione até aqui”. Não tente entender copie e cole.

```
import pandas as pd

titulos =
['NOME','MEDIDA','Local','VAR1','VAR2','VAR3','VAR4','VAR5','var6','var7']

tabela = [

   ['Nome1', 271, 752, 'Local 1', 524, 641, 484, 949, 289, 609],

   ['Nome2', 810, 475, 'Local 2', 724, 905, 500, 160, 790, 636],

   ['Nome3', 433, 750, 'Local 3', 447, 970, 489, 736, 344, 933],

   ['Nome4', 509, 367, 'Local 4', 439, 622, 194, 542, 317, 714],
```

```
    ['Nome5', 440, 669, 'Local 5', 489, 573, 973, 551, 876, 115],
    ['Nome6', 355, 795, 'Local 6', 818, 181, 572, 162, 581, 677],
    ['Nome7', 688, 684, 'Local 7', 207, 180, 594, 412, 125, 107],
    ['Nome8', 252, 826, 'Local 8', 320, 475, 742, 136, 123, 567],
    ['Nome9', 429, 369, 'Local 9', 812, 582, 610, 357, 297, 851],
    ['Nome10',739, 820, 'Local 10', 550, 394, 365, 786, 409, 278],
    ['Nome11', 725, 581, 'Local 11', 821, 812, 716, 644, 396, 455],
    ['Nome12', 281, 150, 'Local 12', 509, 185, 243, 334, 839, 909],
    ['Nome13', 540, 996, 'Local 13', 239, 713, 348, 361, 875, 787],
    ['Nome14', 825, 324, 'Local 14', 375, 301, 178, 622, 788, 709],
    ['Nome15', 781, 762, 'Local 15', 903, 644, 404, 217, 776, 804],
    ['Nome16', 319, 696, 'Local 16', 682, 988, 223, 449, 169, 872],
    ['Nome17', 420, 739, 'Local 17', 347, 758, 478, 532, 497, 771],
    ['Nome18', 735, 352, 'Local 18', 145, 798, 346, 959, 950, 268],
    ['Nome19', 795, 534, 'Local 19', 110, 273, 300, 699, 426, 104],
    ['Nome20', 875, 944, 'Local 20', 126, 603, 351, 678, 760, 725],
    ['Nome21', 592, 382, 'Local 21', 189, 536, 939, 944, 882, 593],
    ['Nome22', 746, 653, 'Local 22', 271, 933, 747, 671, 688, 944],
    ['Nome23', 382, 601, 'Local 23', 948, 772, 551, 485, 828, 507],
    ['Nome24', 387, 618, 'Local 24', 354, 609, 458, 980, 604, 681],
    ['Nome25', 841, 511, 'Local 25', 365, 973, 813, 765, 546, 930],
    ['Nome26', 813, 470, 'Local 26', 964, 700, 535, 456, 518, 511],
    ['Nome27', 101, 117, 'Local 27', 905, 725, 401, 213, 175, 348],
    ['Nome28', 949, 274, 'Local 28', 380, 664, 217, 653, 421, 380],
    ['Nome29', 566, 162, 'Local 29', 185, 944, 222, 839, 646, 871],
    ['Nome30', 788, 489, 'Local 30', 944, 119, 814, 236, 622, 782]
]
tabela = pd.DataFrame(tabela, columns=titulos)
# Exibir a tabela
print(tabela)
```

#selecione até aqui

Agora caso esteja usando um IDE Offline precisa ter a biblioteca “pandas” instalado, cada programa tem seu método.

Copia as informações abaixo desta linha até “#selecione até aqui”. Não tente entender copie e cole.

```
import pandas as pd


# Lendo os dados da planilha do Excel
caminho_arquivo = 'caminho/para/seu/arquivo/dados.xlsx'  # Defina o caminho correto para o seu arquivo
tabela = pd.read_excel(caminho_arquivo)


print(tabela)


#selecione até aqui
```

**Observação:** lembre de alterar o caminho do endereço da planilha, caso a planilha tenha mais de uma aba e deseja escolher uma diferente faça conforme abaixo:

```
tabela_df = pd.read_excel(caminho_arquivo, sheet_name=nome_aba)
```

Pronto temos as informações se você roda o código vai entender melhor as informações.

| | NOME | MEDIDA | Local | VAR1 | VAR2 | VAR3 | VAR4 | VAR5 | var6 | var7 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | Nome1 | 271 | 752 | Local 1 | 524 | 641 | 484 | 949 | 289 | 609 |
| 1 | Nome2 | 810 | 475 | Local 2 | 724 | 905 | 500 | 160 | 790 | 636 |
| .. | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 98 | Nome99 | 622 | 457 | Local 99 | 774 | 304 | 715 | 330 | 266 | 852 |

99  Nome100    950    672  Local 100   349   808   958   778   860   556

[100 rows x 10 columns]

**Comandos:**

A biblioteca panda oferece uma ampla gama de funções e métodos para manipular dados. Alguns dos comandos mais simples e comuns incluem:

1. **`pd.read_csv()`**: Este comando é usado para ler dados de um arquivo CSV e carregá-los em um DataFrame do pandas.

   Exemplo:

   ```python
   import pandas as pd
   df = pd.read_csv('dados.csv')
   ```

2. **`DataFrame.head()` e `DataFrame.tail()`**: Esses métodos são usados para visualizar as primeiras ou últimas linhas do DataFrame.

   Exemplo:

   ```python
   print(df.head())  # Exibe as primeiras 5 linhas do DataFrame
   print(df.tail())  # Exibe as últimas 5 linhas do DataFrame
   ```

3. **`DataFrame.shape`**: Este atributo retorna uma tupla que representa a forma (número de linhas e colunas) do DataFrame.

   Exemplo:

   ```python
   print(df.shape)  # Retorna (número_de_linhas, número_de_colunas)
   ```

4. **`DataFrame.describe()`**: Este método calcula estatísticas descritivas para todas as colunas numéricas do DataFrame, como média, mediana, mínimo, máximo e quartis.

   Exemplo:

   ```python
   ```

print(df.describe())

```

5. **`DataFrame.info()`**: Este método fornece informações sobre o DataFrame, incluindo o tipo de dados de cada coluna e se há valores ausentes.

Exemplo:

```python

print(df.info())

```

6. **`DataFrame.groupby()`**: Este método é usado para agrupar os dados com base em uma ou mais colunas e aplicar funções de agregação.

Exemplo:

```python

df_grouped = df.groupby('coluna_de_interesse').mean()

```

Esses são apenas alguns dos comandos mais simples e comuns que você pode usar para começar a trabalhar com pandas. Existem muitos outros métodos e funções disponíveis para realizar tarefas mais avançadas de manipulação e análise de dados.

Outros:

Tabela.loc[x] = Traz os dados de uma linha específica;

Tabela.iloc[x] = ?

Tabela.iloc[x][y] = ?
```

## EXERCÍCIOS COM RESPOSTAS:

### Strings:

1. **Exercício:** Crie uma string com seu nome e imprima-a.

   **Resposta:**

   ```python
   nome = "João"
   print(nome)
   ```

2. **Exercício:** Crie uma string com uma mensagem e imprima seu comprimento.

   **Resposta:**

   ```python
   mensagem = "Olá, mundo!"
   print(len(mensagem))
   ```

3. **Exercício:** Concatene duas strings e imprima o resultado.

   **Resposta:**

   ```python
   saudacao = "Olá"
   nome = "Maria"
   mensagem = saudacao + ", " + nome
   print(mensagem)
   ```

4. **Exercício:** Converta uma string para letras minúsculas.

   **Resposta:**

   ```python
   texto = "Exemplo de TEXTO"
   texto_minusculo = texto.lower()
   print(texto_minusculo)
   ```

5. **Exercício:** Substitua parte de uma string por outra.

**Resposta:**

```python
frase = "Python é incrível!"
nova_frase = frase.replace("Python", "Java")
print(nova_frase)
```

### Variáveis:

1. **Exercício:** Atribua um número inteiro a uma variável e imprima seu valor.

   **Resposta:**

   ```python
   numero = 10
   print(numero)
   ```

2. **Exercício:** Atribua um número decimal a uma variável e imprima seu valor.

   **Resposta:**

   ```python
   decimal = 3.14
   print(decimal)
   ```

3. **Exercício:** Atribua uma string a uma variável e imprima seu valor.

   **Resposta:**

   ```python
   texto = "Olá, mundo!"
   print(texto)
   ```

4. **Exercício:** Atribua um valor booleano a uma variável e imprima seu valor.

   **Resposta:**

   ```python
   booleano = True
   print(booleano)
```

```

5. **Exercício:** Atribua o resultado de uma expressão aritmética a uma variável e imprima seu valor.

**Resposta:**

```python
resultado = 5 + 3
print(resultado)
```

### Manipulação de String:

1. **Exercício:** Separe uma string em uma lista de palavras.

**Resposta:**

```python
frase = "Python é uma linguagem de programação"
lista_palavras = frase.split()
print(lista_palavras)
```

2. **Exercício:** Conte quantas vezes uma determinada letra aparece em uma string.

**Resposta:**

```python
texto = "Python é uma linguagem de programação"
contagem = texto.count('a')
print(contagem)
```

3. **Exercício:** Verifique se uma string começa com uma determinada subtring.

**Resposta:**

```python
texto = "Python é uma linguagem de programação"
print(texto.startswith("Py"))
```
```

4. **Exercício:** Verifique se uma string termina com uma determinada subtring.

   **Resposta:**

   ```python
   texto = "Python é uma linguagem de programação"
   print(texto.endswith("ção"))
   ```

5. **Exercício:** Junte uma lista de strings em uma única string.

   **Resposta:**

   ```python
   lista = ["Python", "é", "uma", "linguagem", "de", "programação"]
   frase = ' '.join(lista)
   print(frase)
   ```

### Fatiamento de String:

1. **Exercício:** Imprima os primeiros três caracteres de uma string.

   **Resposta:**

   ```python
   texto = "Python"
   print(texto[:3])
   ```

2. **Exercício:** Imprima os últimos três caracteres de uma string.

   **Resposta:**

   ```python
   texto = "Python"
   print(texto[-3:])
   ```

3. **Exercício:** Imprima uma substring de uma string.

   **Resposta:**

   ```python
   ```

```
texto = "Python é incrível!"
print(texto[7:9])
```

4. **Exercício:** Imprima uma substring pulando de dois em dois caracteres.
   **Resposta:**
   ```python
   texto = "Python é incrível!"
   print(texto[::2])
   ```

5. **Exercício:** Inverta uma string.
   **Resposta:**
   ```python
   texto = "Python"
   print(texto[::-1])
   ```

### Números (Operações aritméticas, comparação, lógico):

1. **Exercício:** Faça uma operação de adição entre dois números.
   **Resposta:**
   ```python
   soma = 5 + 3
   print(soma)
   ```

2. **Exercício:** Verifique se um número é maior que outro.
   **Resposta:**
   ```python
   maior = 10 > 5
   print(maior)
   ```

3. **Exercício:** Faça uma operação de multiplicação entre dois números.
   **Resposta:**

```python
multiplicacao = 4 * 7
print(multiplicacao)
```

4. **Exercício:** Verifique se um número é igual a outro.
   **Resposta:**
   ```python
   igual = 3 == 3
   print(igual)
   ```

5. **Exercício:** Faça uma operação lógica usando os operadores 'and' ou 'or'.
   **Resposta:**
   ```python
   resultado = (5 > 3) and (4 < 6)
   print(resultado)
   ```

### Lista:

1. **Exercício:** Crie uma lista com alguns números e imprima-a.
   **Resposta:**
   ```python
   numeros = [1, 2, 3, 4, 5]
   print(numeros)
   ```

2. **Exercício:** Adicione um elemento ao final de uma lista e imprima-a.
   **Resposta:**
   ```python
   lista = [1, 2, 3]
   lista.append(4)
   print(lista)
   ```

3. **Exercício:** Remova um elemento de uma lista e imprima-a.

   **Resposta:**

   ```python
   lista = [1, 2, 3, 4]
   lista.remove(3)
   print(lista)
   ```

### Dicionários:

1. **Exercício:** Crie um dicionário com pares chave-valor representando a relação entre nomes e idades, e imprima-o.

   **Resposta:**

   ```python
   idade = {'Maria': 25, 'João': 30, 'Ana': 20}
   print(idade)
   ```

2. **Exercício:** Acesse o valor associado a uma chave específica em um dicionário e imprima-o.

   **Resposta:**

   ```python
   idade = {'Maria': 25, 'João': 30, 'Ana': 20}
   print(idade['João'])
   ```

3. **Exercício:** Adicione um novo par chave-valor a um dicionário existente e imprima-o.

   **Resposta:**

   ```python
   idade = {'Maria': 25, 'João': 30, 'Ana': 20}
   idade['Pedro'] = 28
   print(idade)
   ```

4. **Exercício:** Remova um par chave-valor de um dicionário existente e imprima-o.

**Resposta:**

```python
idade = {'Maria': 25, 'João': 30, 'Ana': 20}
removido = idade.pop('João')
print("Removido:", removido)
print("Dicionário atualizado:", idade)
```

5. **Exercício:** Verifique se uma chave específica está presente em um dicionário e imprima o resultado.

**Resposta:**

```python
idade = {'Maria': 25, 'João': 30, 'Ana': 20}
presente = 'Maria' in idade
print(presente)
```