**Conhecendo a Linguagem Python**

* Tipos de dados: definem as características e comportamentos de um valor (objeto).
* Tipos built-in são:
* **Texto – str (string)**

Utilizado para armazenar sequências de caracteres. Strings são imutáveis, o que significa que uma vez criadas, suas partes internas não podem ser alteradas. Strings podem ser delimitadas por aspas simples (' ') ou duplas (" ").

* **Numérico – int, float, complex**

**int (inteiro)**: Representa números inteiros, positivos ou negativos, sem casas decimais. Não há limite para o tamanho de um inteiro em Python, exceto pela memória disponível. Ex: 42

**float (ponto flutuante)**: Utilizado para representar números com casas decimais. Este tipo de dado é uma aproximação dos números reais. Ex: 3.14

**complex (número complexo)**: Utilizado para representar números complexos, que possuem uma parte real e uma parte imaginária. A parte imaginária é denotada com um 'j' no final. Ex: 1 + 2j

* **Sequência – list, tuple, range**

**list (lista)**: Uma coleção ordenada e mutável de elementos. Listas são definidas por colchetes [ ] e podem conter elementos de diferentes tipos. Ex: [1, 2, 3, "a", "b", "c"]

**tuple (tupla)**: Similar a uma lista, mas é imutável. Tuplas são definidas por parênteses ( ). Ex: (1, 2, 3, "a", "b", "c")

**range (intervalo)**: Representa uma sequência de números gerados dentro de um intervalo especificado. É comumente usado em loops. Ex: range(1, 10)

* **Mapa – dict (dicionário)**

Um conjunto de pares chave-valor. Dicionários são definidos por chaves { } e são mutáveis. Ex: {"name": "Alice", "age": 25, "city": "New York"}

* **Coleção – set, frozenset**

**set (conjunto)**: Uma coleção não ordenada de elementos únicos. Sets são definidos por chaves { }, mas com elementos separados por vírgulas. Ex: {1, 2, 3, 4, 5}

**frozenset (conjunto congelado)**: Similar ao set, mas é imutável. Pode ser criado utilizando a função frozenset(). Ex: frozenset([1, 2, 3, 4, 5])

* **Booleano – bool (booleano)**

Representa um valor lógico que pode ser True ou False. Em python o tipo booleano é uma subclasse de int, uma vez que qualquer número diferente de 0 representa verdadeiro e 0 representa falso

* **Binário – bytes, bytearray, memoryview**

**bytes (bytes)**: Uma sequência imutável de números inteiros no intervalo de 0 a 255. Utilizado para representar dados binários. Ex: b"hello"

**bytearray (array de bytes)**: Semelhante a bytes, mas mutável. Ex: bytearray(b"hello")

**memoryview (visão de memória)**: Permite acesso a uma sequência de bytes sem copiar os dados. É útil para manipulação de grandes volumes de dados binários. Ex: memoryview(b"hello")

* **Modo interativo**: ir no terminal e colocar “python” e dar enter. E pra sair escrever “exit()”. Outra forma é colocar “python -i” + o nome do arquivo que vc vai querer abrir
* Função **dir**: Quando chamada sem argumentos, ela retorna a lista dos nomes no escopo local atual. Quando chamada com um argumento, ela retorna a lista dos nomes dos atributos e métodos do objeto passado como argumento, métodos que podem ser usados no objeto, algo assim.
* Dir()
* Dir(100)
* **Help**: Invoca o sistema de ajuda integrado. É possível fazer buscas em modo interativo ou informar por parâmetro qual o nome do módulo, função, classe, método ou variável. é usada para exibir a documentação de um objeto, módulo ou função. Ela é extremamente útil para entender o que uma função ou módulo faz, quais argumentos aceita e qual é o propósito de um objeto. Exemplo:
* Help()
* Help(100)
* Help(my\_string.upper)
* O python consegue reconhecer o tipo da variável na própria atribuição, não sendo necessário você definir se a variável é int, bool, float.
* Em python é possível atribuir várias variáveis em uma única linha, ex:

nome, idade = "Guilherme", 28

* Não existe uma palavra reservada para informar ao interpretador que um certo valor é constante, diferentes de algumas linguagens como Java e C. Mas em python temos a convenção de definir uma variável como constante colocando seu nome em maiúsculo. Ex: DEBUG = True
* Convenções python:
* Snake case: Os nomes das variáveis devem ser separados por \_ ex: Primeira\_Compra
* Escolher nomes sugestivos, para melhor legibilidade do algoritmo
* Nome de constantes todo em maiúsculo
* Conversão de tipos: as vezes é necessário por exemplo converter uma variável do tipo string que armazena número para int ou float
* Inteiro para float:
* Preco = 10
* Preco = float(preco)
* Print(preco)
* >>>> 10.0
* Ao dividir um número inteiro por outro valor também vai ocorrer a conversão de inteiro para float (ex: 5/10 = 5.0), a menos que vc use 2 barras, dai o numero continua como inteiro (5//10 = 5)
* De float para int é semelhante: preco = int(preco). Nesse caso ele simplesmente corta os decimais, nn arredonda, se o número for 10.9 por exemplo fica só 10
* Numérico para string:
* str(preco)
* para converter string para número é só usar int(preco) ou float(preco), mas pra que isso ocorra é necessário que o dado contido na variável seja numérico
* Concatenação:
* Texto = **f”idade {idade} preco {preco}”**
* Print(texto)
* >>> idade 28 preco 10.5
* Comando para exibir o tipo da variável: **type()**
* Comandos de entrada e saída: **input/print**
* Print(nome, sobrenome) >>> Thyago Gomes
* Print(nome, sobrenome, **end=”.../n”)** >>> Thyago Gomes... – com isso eu termino com 3 pontos e insiro uma quebra de linha (/n)
* Print(nome, sobrenome, **sep=”#”)** >>> Thyago#Gomes - separador

**Tipos de Operadores com Python**

* **Operadores aritméticos**
* **+ - / \***
* **%**  - resto da divisão, módulo. Ex: 10%3 = 1
* **//** - divisão inteira. Obs: se o número q vc for dividir for float, ele vai continuar como float, apesar de ficar inteiro, ex: 25.5//2 = 12.0
* **\*\*** - exponenciação
* Ordem de procedência: parênteses, exponenciação, multiplicação e divisão, somas e subtrações
* Função **math.sqrt** – obtem raiz quadrada de um número. Mas pra isso é necessário importar o módulo math:
* Import math
* Print(math.sqrt(16)) >>> saída: 4.0
* Outra forma de obter a raiz quadrada de um número é elevá-lo a potência de 0.5. Ex: 16\*\*0.5 = 4
* Os parênteses podem ser usados pra mudar a ordem de uma operação, ou podem ser usados pra deixar mais claro e legível qual a ordem em que o cálculo vai ser resolvido (apesar de q acho q pra alguns poderia parecer redundante)
* **Operadores de comparação**
* **< > == != <= >=**
* **Operadores de atribuição**
* **=** - sinal de atribuição simples, atribui um valor a uma variável
* **+=** - soma o valor atribuído com a própria variável.

Ex: saldo += 200, seria o mesmo que: saldo = saldo + 200.

Obs: não funciona dentro do print – print (saldo += 200) da erro, tem q atribuir fora pra depois printar a variável

* Semelhante a esse também temos: **-= \*= /= //= %= \*\*=**
* **Operadores lógicos**
* **and**, **or**, **not**
* Algumas observações:
* Dentro de um parâmetro lógico sequências vazias são reconhecidas como falso, da mesma forma string vazias, e acho q variáveis vazias também; já se tiver preenchido são reconhecidas como verdadeiras. Ex:
* contatos = []
* print(not contatos)
* print(not "saque")
* print(not "")
* Retorna: True, False, True
* Parenteses também podem ser usados pra definir a ordem que as comparações serão feitas, ou simplesmente pra facilitar a leitura pra bater o olho e entender quais operações serão feitas primeiro. Ex:

Saldo >= saque **and** saque <= limite **or** conta\_especial **and** saldo >= saque

Pra ficar mais legível posso colocar:

(Saldo >= saque **and** saque <= limite) **or** (conta\_especial **and** saldo >= saque)

As duas formas retornarão o mesmo

* **Operadores de Identidade**
* Os operadores de identidade em Python são usados para comparar objetos, verificando se eles são o mesmo objeto na memória.
* Existem dois operadores de identidade em Python: **is**  e **is not**
* **Is:**
* Verifica se dois objetos referenciam o mesmo objeto na memória
* Ex:

A = 10

B = 5

C = 10

print (a **is** b) >>> False

print (a **is** c) >>> True

* **Is not**: é o inverso do is
* Diferença entre is e ==

**is** Compara se os dois operandos referenciam o mesmo objeto (identidade do objeto).

**==** Compara se os valores dos dois operandos são iguais (igualdade de valor).

* **Operadores de Associação**
* **In**  e **not in**
* Ex:

Curso = “Curso de Python”

Frutas = [“laranja”, “uva”, “limão”]

Saques = [1500, 100]

“Python” **in** curso >>> True

“python" **in** Curso >>> False

“maçã” **not in** frutas >>> True

200 **in** saques >>> False

**Estruturas Condicionais e de Repetição**

* **If, elif, else**: lembrar de sempre colocar dois pontos “:” após a sentença, e a indentação é necessária para o programa compreender o que está dentro de cada estrutura
* **If ternário**:

Status = “Sucesso” **if** saldo >= saque **else** “Falha”

É uma forma de simplificar, pra casos curtos.

* As duas principais estruturas de repetição em Python são o loop **for** e o loop **while**
* **For:**
* for <var> in range(<inicial>, <final>, <incremento>):

Instrução (ões)

* Porém dentro do parênteses vc pode colocar só o valor final, nesse caso o inicial será considerado 0, e o incremento será 1
* Um exemplo:

For i in **range** (1, 6, 1):

Print (i)

Saída: 1 2 3 4 5

Obs: note q o valor 6 nn será printado, ele mostrará somente os valores abaixo do 6

* Também é possível iterar **strings**:

Palavra = “python”

for letra in palavra:

print(letra)

Saída >>> p y t h o n

* Vc pode colocar um “**else**” no final pra definir um comando que será executado depois do for acabar ex:

For blablabla

Comando

Else:

Comando

* Observe o seguinte exemplo:

frase = "Lógica de Programação e Algoritmos"

for i in range(0, len(frase), 1):

print(frase[i], end="")

colocando **“len(frase)”** pra referenciar o tamanho da string, e

**“end=”””** pra fazer com que o programa print cada letra ao lado da outra

E **[i]** serve como um índice, para acessar elementos específicos da string. Frase[0] retorna L, e assim por diante

* Também é possível iterar **listas**:

frutas = ['maçã', 'banana', 'cereja']

for fruta in frutas:

print(fruta)

* Você pode iterar sobre as chaves, valores ou itens (pares chave-valor) de um **dicionário**:

aluno\_notas = {'Ana': 8.5, 'Pedro': 7.0, 'João': 9.0}

for aluno, nota in aluno\_notas.items():

print(f'{aluno}: {nota}')

* **While:**
* É usado quando não se tem certeza do numero de vezes que o comando vai ser repetido, diferente do for que é bem delimitado
* O loop while continua executando enquanto a condição especificada for verdadeira

contador = 0

while contador < 5:

print(contador)

contador += 1

Saída: 0 1 2 3 4

* Um exemplo comum é usar while para continuar pedindo uma entrada do usuário até que a entrada seja válida:

senha = ''

while senha != '1234':

senha = input('Digite a senha: ')

print('Acesso concedido')

* A
* **Break**: é uma palavra de parada, irá parar a estrutura de repetição e continuar o programa
* **Continue**: serve pra pular uma condição específica, pular uma repetição dentro do laço, por exemplo se colocar pra o programa escrever sequência de 1-10, e se o número for par continue, ele vai escrever só os impares

**Strings e Fatiamento**

* Maiúscula, minúscula e título:
* curso = “PYthOn”
* Print(**curso.upper()**) >>> PYTHON
* Print(**curso.lower())** >>> python
* Print(**curso.title())** >>> Python
* Eliminando espaços em brancos:
* curso = “ Python “
* Print(**curso.strip())** >>> “Python”
* Print(**curso.lstrip())** >>> “Python “
* Print(**curso.rstrip())** >>> “ Python”
* Junções e centralização:
* Curso = “python”
* Print(**curso.center(10, “#”))** >>> “##python##”

Faz com que a palavra fique centralizada, e com que tenha 10 caracteres, preenchendo o que falta com #. O segundo argumento dessa função center, o “#” é opcional, se vc nn coloca-lo o programa irá acrescentar espaços em branco. É bom pra fazer um menu bonitinho

* Print(**“.”.join(curso))** >>> “p.y.t.h.o.n”

O join vai passer item a item da string e a cada item ele vai acrescentar o caractere que vc colocou, nesse caso um ponto “.”. É bom as vezes pra usar com listas

* **Interpolação de variáveis**
* Utilizando “%” – não é mais recomendado utilizar hj em dia
* Print("Olá, me chamo %s. Eu tenho %d anos de idade, trabalho com %s e estou matriculado no curso de %s.” % (nome, idade, profissão, linguagem))
* Método ‘str.format’:
* Print("Olá, me chamo {}. Eu tenho {} anos de idade, trabalho como {} e estou matriculado no curso de {}.”.format(nome, idade, profissão, linguagem))
* Também é possível definir a ordem em q as strings vão aparecer, é útil caso vá por exemplo usar alguma variável várias e várias vezes:
* Print("Olá, me chamo {3}. Eu tenho {2} anos de idade, trabalho como {1} e estou matriculado no curso de {0}.”.format(linguagem, profissão, idade, nome))
* **F-strings:** a mais moderna e mais utilizada hoje:
* Print(f”Olá, me chamo {nome}. Eu tenho {idade} anos de idade, trabalho como {profissão} e estou matriculado no curso de {linguagem}.”)
* **Formatação**:

Print(f”Valor de PI: {**PI:.2f**}”) >>>> “Valor de PI: 3.14”

Print(f”Valor de PI: {**PI:10.2f**}”) >>>> “Valor de PI: 3.14”

F no final é de “float”

O 10 antes do ponto é a quantidade de espaços que vão ser adicionados à esquerda pelo que entendi

* **Fatiamento de Strings:**
* É uma técnica utilizada utilizada para retornar substrings (partes da string original), informando o início (start - inclusivo), o fim (stop - exclusivo) e o passo (step - tamanho do salto entre os índices)
* Obs – o primeiro índice é 0, por isso o ultimo índice acaba sendo exclusivo
* **[start: stop[, step]]**
* Exemplos:
* nome = “guilherme arthur de carvalho”
* nome [0] >>> “g”
* nome [:9] >>> Guilherme
* nome [10:] >>> arthur de carvalho
* nome [10:16] >>> arthur
* nome [10:16:2] >>> atu
* nome [:] >>> retorna a variável inteira – Guilherme Arthur de carvalho
* nome [::-1] >>> espelha a variável – ohlavrac ed ruhtra emrehliug
* **Strings de múltiplas linhas (strings triplas):**
* São definidas informando 3 aspas simples ou duplas durante a atribuição. Elas podem ocupar várias linhas do código, e todos os espaços em branco são incluídos na string final
* Ex:

print (f"""Olá meu nome é {nome}

tenho 22 anos

gosto de jogar""")

* **Muito importante na criação de menus**