Seção 2: Python Básico

▼ Comentários de código em Python

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/94ba7a2f-309c-4d3e-9666-5389262e6209/aula1.py

- Comentários óbvios devem ser evitados. O comentário é útil para casos de linhas complexas em que uma lógica mais elaborada foi usada;
- Usado para omitir e não executar partes do código;
- Comentário de uma linha só é usado do símbolo #. Para comentários de múltiplas linhas é usado três aspas duplas (esse método não é um comentário em si, é uma *docstring*, porém pode ser usado como comentário).

```
print('Linha 1')
print('Linha 2')
# Falando sobre do que se trata essa linha
print('Linha 3')
print('Linha 4')  # Isso é um comentário
"""
Comentário de multiplas
linhas
nessa parte do
código
"""
print('Linha 5')
```

▼ O comando print

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/89b8f94e-d417-4bf9-a67a-686e1f7ecb4f/aula2.py

- Todo comando que apresenta *( ) "abre a fecha parênteses"* é uma função. A função print mostra o que se pede na tela;
- sep=: você define como os argumentos passados serão separados. Por padrão o python separa com espaços, mas com esse argumento é possível alterar.
- Por padrão *print's* seguidos são sempre em novas linhas, mas com o argumento *end*= também é possível alterar isso, fazendo com que o próximo print ocorra na linha atual.

```
print(123456)
print('Conhecendo', 'o', 'separador', sep='-')
```

```
print('Testando', end='~~')
print('Como se mostra dois prints na mesma linha')

# 999.888.777-66
print(999, 888, 777, sep='.', end='-')
print(66)
```

▼ Strings e aspas em Python

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/cb8a955a-0f39-4668-b7aa-f15cde63b65b/aula3.py

- Python reconhece tanto aspas simples como aspas duplas;
- Para usar aspas em alguma coisa no print, é necessário ou usar tipos diferentes de aspas ou um caractere de escape, que no caso do python é a barra invertida ‘ \ ‘;
- É possível que algumas coisas dentro do print sejam executáveis. Caso não queira que isso aconteça, antes das aspas que inicia a string é acrescentado um ‘r’, isso fará que tudo que está entre as aspas seja uma string;

```
# str - string

print('Alguma coisa')
print("Aspas duplas")
print('Alguma coisa entre "aspas"')
print("Outra coisa entre 'aspas'")
print('Caractere de \'escape\'')
```

▼ Tipos de dados “primitivos”

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/6873e2a9-97a5-4906-be0f-cf2cd47589e9/aula4.py

- Para ter certeza de qual tipo algum dado possui, a função *type* pode ser usada;
- Quando tipos são convertidos para booleano, eles sempre vão receber o valor True, com exceção de dados vazios e o número 0;

```
"""
Tipos de dados:
str - string (textos, coisas dentro de aspas)
int - inteiro (números positivos ou negativos inteiros)
float - real (qualquer número real positivo ou negativo, separado por PONTO)
bool - booleano/lógico (True ou False)
"""

print('Andre', type('Andre'))
print(10, type(10))
print(10.5, type(10.5))
print(10 == 10, type(10 == 10))

print('Andre Prasel', type('Andre Prasel'))
```

```
print(20, type(20))
print(1.91, type(1.91))
print(20 > 18, type(20 >= 18))
```

▼ **Operadores aritméticos**

- O operador de multiplicação também funciona como símbolo de repetição de caracteres, ou seja, ao multiplicar uma string por um número inteiro, essa string irá se repetir *n* vezes;
- O operador de soma ao ser usado em strings realiza a concatenação dessas strings.

▼ **Variáveis**

- Deve iniciar com letra, pode conter números, separar por _ (underline), letras minúsculas.

▼ **Entrada de dados**

- Por padrão, a função *input* converte todos os valores digitados para *string*, assim, se for necessário fazer alguma operação aritmética não será possível até que haja a conversão dos dados.

▼ **Pass e Ellipsis como placeholders**

- Pedaços de códigos normalmente dentro de condicionais que servem para pular essa parte do código. Isso é feito para que posteriormente algum código seja aplicado nesse ponto, mas que no momento não existe (isso evita algum tipo de erro do interpretador). Para fazer isso basta usar o comando *pass* ou . . . (três pontos).

▼ **Formatando valores em python**

- : . (número)f - quantidade de casas decimais (float)
- : (caractere) (> ou < ou ^) (quantidade) (tipo - s, d ou f)
  - > - caracteres na esquerda do dado
  - < - caracteres na direita do dado
  - ^ - centraliza o dado entre os caracteres.

▼ **While/Else**

- Diferente de outras linguagens de programação, o python pode conter um bloco Else fora de um While. Esse bloco só será executado se a condição analisada no While for falsa. Caso o While for quebrado, o Else não será executado.

▼ **Listas**

- Fatiamento padrão, semelhante ao de strings;
- Podem ser alteradas;
- É possível fazer concatenação;
- *extend*: usado para estender uma lista. Pode receber vários tipos de valores, inclusive outras listas;
- *append*: insere um novo dado no final da lista;
- *insert*: insere um novo dado na posição desejada na lista (indicar o índice);
- *pop*: exclui o último dado da lista;

- *del*: exclui os dado por índice (é possível usar fatiamento);
- *max e min*: pega o valor máximo ou mínimo de uma lista;

▼ **FOR / ELSE**

- Assim como o While, o For também suporta um laço Else;

▼ **Desempacotamento de listas**

- Forma de atribuir a cada variável um valor de uma lista, seguindo a sequência dos índices;

```
nomes = ['André', 'Helena', 'Nikito']
nome1, nome2, nome3 = nomes
print(nome2)

>>> Helena

print(nome3)

>>> Nikito
```

Caso não seja informado uma variável para cada item, é necessário criar uma variável com * (asterisco), isso faz com que os dados restantes fiquem guardados nessa variável criada por meio de uma lista.

```
lista = ['André', 'Helena', 'Nikito', 1, 2, 3, 4, 5, 6, 7, 8, 9]
nome1, nome2, nome3, *numeros = lista

print(nome3, numeros)

>>> Nikito [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Se outra variável for criada depois da variável com asterisco, essa sempre vai receber o último valor da lista original.

▼ **Trocando o valor entre variáveis em Python**

Isso é feito para inverter os valores entre variáveis, sem a necessidade de variáveis de auxílio.

```
x = 10
y = 'Andre'

print(f'x={x} e y={y}')

>>> x=10 e y=Andre

x, y = y, x  #aqui x está recebendo o valor de y e y está recebendo o valor de x.

print(f'x={x} e y={y}')

>>> x=Andre e y=10


# é possível fazer isso com mais de duas variáveis.
```

▼ **Expressão condicional**

```
<expressao1> if <condicao> else <expressao2>
```

Primeiro, a condição é avaliada (ao invés de `expressao1`), se a condição for verdadeira, `expressao1` é avaliada e seu valor é retornado; caso contrário, `expressao2` é avaliada e seu valor retornado.

Com base no teu exemplo, o código fica assim:

```
x = 10
print ("par" if x % 2 == 0 else "impar")
```

É possível fazer isso com o operador OR. Exemplo:

```
nome = input('Qual o seu nome? ')
print(nome or 'Você não digitou nada :(')

# o que está acontecendo aqui é que se o valor de nome for verdadeiro, irá executar a primeira expressão.
# Caso contrário, a segunda expressão irá ser executada.
```