

Relatório - Linguagem de Programação Python (I)

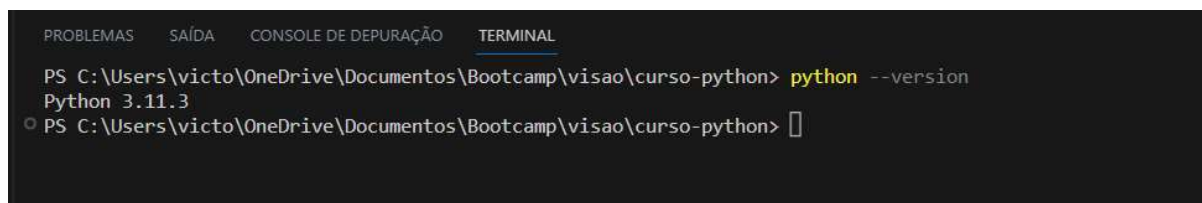
Paulo Victor Sousa de Almeida

1 – Python.

Python é uma linguagem de programação de alto nível, interpretada e de propósito geral. Foi criada por Guido van Rossum e lançada pela primeira vez em 1991. Python se destaca por sua sintaxe clara e legível, o que a torna uma linguagem muito acessível para iniciantes, além de ser utilizada em diversos domínios de desenvolvimento de software.

1.1 Python 3.

Python 3 é a versão mais recente e atualizada da linguagem de programação Python. Foi lançado em dezembro de 2008. Neste curso foi utilizado a versão 3 do Python, sendo a versão exata configurada no meu ambiente de desenvolvimento e a 3.10.11.



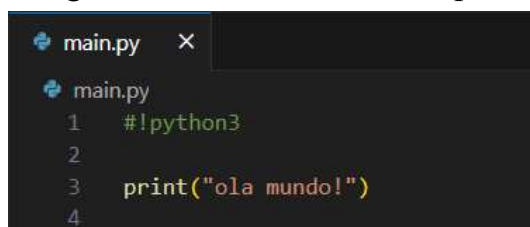
```
PROBLEMAS  SAÍDA  CONSOLE DE DEPURACÃO  TERMINAL

PS C:\Users\victo\OneDrive\Documentos\Bootcamp\visao\curso-python> python --version
Python 3.11.3
○ PS C:\Users\victo\OneDrive\Documentos\Bootcamp\visao\curso-python> 
```

2 – Termos básicos.

2.1 Shebang.

O Shebang “#!” É o termo utilizado para declarar o uso de um compilador específico, usado em Perl, Python, Shell Script ou qualquer outro script. Sendo obrigatoriamente utilizado na primeira linha do arquivo.



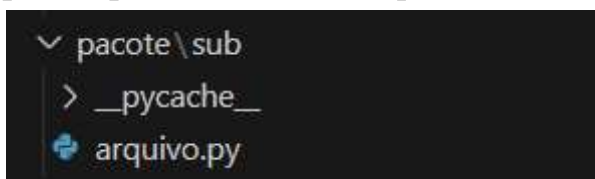
```
main.py x
main.py
1  #!python3
2
3  print("ola mundo!")
4
```

2.2 Pep8.

O PEP8 é um guia de estilo para programação em Python. Ele define diretrizes e convenções sobre como escrever código Python de forma clara, legível e consistente. O objetivo do PEP8 é tornar o código mais fácil de ler e entender para os programadores, promovendo uma padronização na comunidade Python. Para formatar o código Python basta utilizar o atalho (ALT + SHIFT + F).

3 – Pacotes / Módulos.

Em Python, pacotes são diretórios que contêm módulos Python relacionados. Eles são usados para organizar e estruturar o código em projetos maiores. Um pacote pode conter outros pacotes, além de módulos e subpacotes.



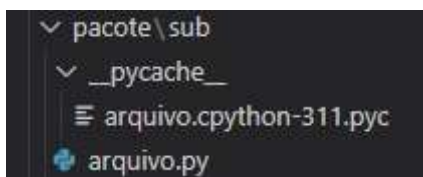
3.1 Import.

Para importar um pacote em Python, você usa a declaração import. Por exemplo, se você tem um pacote chamado 'meu_pacote', pode importá-lo da seguinte forma: import 'meu_pacote'. Depois disso, você pode acessar os módulos e subpacotes do pacote usando a sintaxe de ponto, como 'meu_pacote.modulo1' ou 'meu_pacote.subpacote'.

```
4
5 import pacote.sub.arquivo
6 import tipos.variaveis
7
```

3.2 Pycache.

A pasta "pycache" é um diretório especial gerado pelo interpretador Python para armazenar os arquivos de bytecode compilados (.pyc) durante a execução do programa. Esses arquivos de bytecode são criados para melhorar o desempenho e acelerar a importação de módulos Python.



Quando você executa um programa Python, o interpretador compila o código fonte (.py) em bytecode, uma forma intermediária que é mais rápida de ser interpretada do que o código fonte original. O bytecode é então armazenado nos arquivos .pyc dentro da pasta "pycache".

4 Tipos de Variáveis.

4.1 Tipos Primitivos.

Em Python, existem diversos tipos primitivos embutidos na linguagem. Esses tipos são usados para representar diferentes categorias de dados e fornecer funcionalidades específicas. Aqui estão os tipos primitivos mais comuns em Python.

```
curso-python > tipos > + basicos.py
1 | print(type(1))           # imprime int
2 | print(type(1.1))         # imprime float
3 | print(type('texto'))    # imprime String
4 | print(type(False))      # imprime Bool
5 | print(type(True))       # imprime Bool
```

4.2 Atribuição de Variáveis.

Ao contrário de outras linguagens de programação, em Python, você não precisa declarar explicitamente o tipo da variável. O tipo da variável é inferido a partir do valor atribuído a ela.

```
curso-python > tipos > + variaveis.py > ...
1 | #!python3
2 | a = 3           # Int
3 | b = 4.4         # Float
4 | c = 'Ola'       # String
5 | d = True        # Bool
6 | e = False       # Bool
7 |
```

4.3 F-String.

As f-strings (format strings) são uma forma conveniente e eficiente de formatar strings em Python a partir da versão 3.6. Elas permitem incorporar expressões Python dentro de strings delimitadas por aspas ou apóstrofos precedidos pelo prefixo "f" ou "F".

```
texto = 'Sua idade: '      # Cadeia de Caracteres
idade = 23                 # inteiro
print(f'{texto} {idade}')  # concatena String + inteiro, F
```

```
PS C:\>
Sua idade: 23
```

5 - Listas.

Uma lista é um tipo de dado que representa uma sequência mutável e ordenada de elementos. As listas são uma estrutura de dados muito versátil e útil, e são amplamente utilizadas em Python.

```
curso-python > tipos > lista.py > ...
1  nums = [1,2,3]          # Atribuir Lista
2  print(type(nums))       # Imprime Tipo Lista
3
```

5.1 Métodos da Lista.

5.1.1 Método append.

Adiciona um valor ao final da lista.

```
3
4  nums.append(3) # adiciona o valor 3 ao final da lista
5
```

5.1.2 Método insert.

Insere um valor a um índice

```
12
13  nums.insert(0, -200) # insere o elemento -200 no índice 0 da lista
14
```

5.1.3 Método remove.

remove valor da lista

```
20
21 |  nums.remove(4) # remove o valor 4 da lista
22
```

6 - Tuplas.

Uma tupla é um tipo de dado que representa uma sequência imutável e ordenada de elementos. Ao contrário das listas, as tuplas não podem ser modificadas após a sua criação. Elas são usadas quando você precisa armazenar um conjunto de valores que não deve ser alterado. Para criar uma tupla em Python, você usa parênteses () e separa os elementos por vírgulas.

```
curso-python > tipos > tuplas.py > ...
1  nomes = ('Ana', 'Bia', 'Gui', 'Leo', 'Ana')
2
```

Assim como nas listas, os elementos de uma tupla são indexados. A indexação começa em 0, então o primeiro elemento está no índice 0, o segundo elemento está no índice 1 e assim por diante. Você pode acessar um elemento específico usando seu índice.

```
4
5  print(nomes[0])    # imprime elemento do índice 0
6  print(nomes[1:3])  # imprime elemento do índice 1 ate 2
7  print(nomes[1:-1]) # imprime elemento do índice 1 ate ante penultimo indice
8  print(nomes[2:])   # imprime todos os elemento depois do indice 2
9  print(nomes[:-2])  # imprime todos os elementos antes do antepenultimo indice
10
```

7 - Conjuntos.

um conjunto (set) é um tipo de dado que representa uma coleção não ordenada e mutável de elementos únicos. Os conjuntos são úteis quando você precisa armazenar elementos sem se preocupar com a ordem ou com a ocorrência de duplicatas. Para criar um conjunto em Python, você pode usar chaves {} ou a função set().

```
4
5  conj1 = {1, 2, 3, 3, 3, 3}
6  conj2 = set([4, 5, 6, 6, 6])
7
```

Conjuntos além de não considerarem valores repetidos, eles não permitem acessar elementos a partir de valores indexados.

```
10 print(type({1, 2, 3}))
11
12 print(conj1)
13 print(len(conj1))
14
15 print(conj2)
16 print(len(conj2))
```

```
<class 'set'>
{1, 2, 3}
3
{4, 5, 6}
3
```

6 - Dicionário.

Um dicionário é um tipo de dado que representa uma coleção de pares chave-valor. É uma estrutura de dados extremamente útil quando você precisa associar valores a chaves únicas.

```
1 aluno = {
2     'nome': 'Pedro Henrique',
3     'nota': 9.5,
4     'ativo': True
5 }
6
```

```
prof = dict(nome = 'Fernando', diciplina = 'Matematica', ativo = True)
```

Os dicionários são mutáveis, o que significa que você pode adicionar, modificar ou remover elementos do dicionário.

```
prof['ativo'] = False
```

```
{'nome': 'Fernando', 'diciplina': 'Matematica', 'ativo': True}
{'nome': 'Fernando', 'diciplina': 'Matematica', 'ativo': False}
```

Observe que o valor da variável que era True mudou para False.

7 - Operadores.

7.1 - Operadores Unários.

Operadores unários são operadores que trabalham com apenas um operando. Eles são usados para realizar operações em um único valor ou variável.

```
1  x = 3
2  y = 4
3
4  print(not False)    # Operador de negacao logico / imprime negacao de falso
5  print(not True)     # Operador de negacao logico / Imprime negacao de verdadeiro
6  print(-x)           # Operador de negacao / investe o valor do numero
7  print(--y)           # Operador de negacao / imprime o valor de negacao da negacao
8  print(+3)           # nao existe operador de incremento e decremento
9  print(x is y)       # Operador de identidade / imprime False se for objetos diferentes
10
```

7.2 - Operadores Aritméticos.

Operadores aritméticos são usados para realizar operações matemáticas em valores numéricos.

```
1  x = 10
2  y = 3
3
4  # Operadores binarios, operam em cima de dois operandos, sintaxe infix
5  print(x + y)        # Operador de adicao
6  print(x - y)        # Operador de subtracao
7  print(x * y)        # Operador de multiplicacao
8  print(x / y)        # Operador de divisao
9  print(x % y)        # Operador de resto da divisao
10 print(x // y)       # Operador de divisao inteira
11
```

7.3 - Operadores Relacionais.

Os operadores relacionais são usados para comparar valores e retornar um resultado booleano (True ou False) com base na relação entre os valores.

```

1  x = 7
2  y = 5
3
4  print(x > y)           # Operador maior que
5  print(x >= y)          # Operador maior ou igual que
6  print(x < y)           # Operador menor que
7  print(x <= y)          # Operador menor ou igual que
8  print(x == y)          # Operador de igualdade
9  print(x != y)          # Operador de desigualdade
10

```

7.4 - Operadores de Atribuição

Os operadores de atribuição são usados para atribuir valores a variáveis. Eles permitem que você atualize ou modifique o valor de uma variável com base em uma expressão.

```

1  resultado = 2          # atribuicao simples
2  resultado += 3          # atribuicao aditiva
3  resultado -= 3          # atribuicao subtrativa
4  resultado *= 3          # atribuicao multiplicativa
5  resultado /= 3          # atribuicao divisiva
6  resultado %= 3          # atribuicao modulo
7  resultado **= 3         # atribuicao potencial
8  resultado //= 3         # atribuicao divisao inteira
9

```

7.5 - Operadores Lógicos

Os operadores lógicos são usados para combinar ou manipular valores booleanos (True ou False). Eles permitem realizar operações lógicas em expressões condicionais e tomar decisões com base nas condições.

```

1  b1 = True
2  b2 = False
3  b3 = True
4
5  print(b1 and b2 and b3) # Operador logico E(And)
6  print(b1 or b2 or b3)  # Operador logico OU(Or)
7  print(b1 != b2)        # Pseudo Operador Logico OU Exclusivo(Xor)
8  print(not b1)           # Operador Logico Negacao
9  print(not b2)           # Operador Logico Negacao
10 print(b1 and not b2 and b3) # Multiplos Operadores
11

```


7.6 - Operador Ternário

Em Python, não há um operador lógico ternário como em algumas outras linguagens de programação, é possível obter um comportamento semelhante usando uma expressão condicional (também conhecida como operador condicional) em uma única linha.

```
2 lockdown = False
3 grana = 101
4
5 #status recebe '' se / Resultado se Verdadeiro / Esprecao Logica / Resultado se Falso
6 status = 'em casa' if lockdown or grana <= 100 else 'Uhuuuu'
7
```

8 - Estrutura de Controle

Em Python, o controle de fluxo refere-se às instruções e estruturas que permitem controlar a execução do programa com base em condições ou iterações.

8.1 - if-elif-else

A instrução “if” permite executar um bloco de código se uma determinada condição for verdadeira. A “if-else” executa um bloco de código se a condição for verdadeira e outro bloco de código se a condição for falsa. E a “if-elif-else” verificar múltiplas condições em sequência e executar o bloco de código correspondente à primeira condição verdadeira.

```
2 nota = float(input('Informe Nota: '))
3 comportado = True if input('Comportado: (y/n): ') == 'y' else False
4
5 if nota >= 9 and comportado:
6     print('Parabens!')
7     print('Quadro de Honra')
8 elif nota >= 7:
9     print('Aprovado')
10 elif nota >= 5.5:
11     print('Recuperacao')
12 elif nota >= 3.5:
13     print('Recuperacao + Atividade')
14 else:
15     print('Reprovado')
16
17 print(nota)
```

8.2 - for

A estrutura de repetição "for" é usada para iterar sobre uma sequência de elementos, a cada iteração do loop. O bloco de código indentado abaixo do "for" será executado para cada elemento da sequência.

```
6 for i in range(1, 11):
7     print(i, end=' ')          # imprime de 1 a 10 / end imprime na mesma linha
8 print('')
9
```

8.3 - while

A estrutura de repetição "while" é usada para executar um bloco de código repetidamente enquanto uma determinada condição for verdadeira.

```
x = 10
while x:
    print(x)
    x -= 1
print('Fim')
```

9 - Funções

Para definir uma função devemos usar a palavra-chave "def" em seguida vem o nome da função e parâmetros, o bloco de códigos da função é definido com indentação.

```
1  #!python3
2
3  # Funcao saudacao
4  def saudacao(nome='Pessoa', idade='Idade'):
5      # Bloco de código da funcao
6      print(f'Bom dia {nome}!, Idade: {idade} anos!')
7
```

10 - Função funcional

É o paradigma de programação que trata as funções como argumentos para outras funções e retorna elas como resultados de funções.

```
def soma_parcial(a):  
    def concluir_soma(b):  
        return a + b  
    return concluir_soma  
  
fn = soma_parcial(10) # fn recebe A funcao concluir_soma  
resultado_final = fn(12) # funcao retorna valor da funcao concluir_soma  
print(resultado_final)  
  
# ou  
  
resultado_final = soma_parcial(10)(12)  
print(resultado_final)
```

11 - lambda

A função lambda é uma função que não requer nome, podendo ser em uma só linha e pode ter qualquer argumento.

```
aluno_aprovado = lambda aluno: aluno['nota'] >= 7  
  
aluno_honra = lambda aluno: aluno['nota'] >= 8  
  
obter_nota = lambda aluno: aluno['nota']
```

12 - Orientado a Objeto

Em Python o POO é amplamente suportado e implementado através dos principais conceitos como classes, objetos, atributos, métodos, encapsulamento, herança e polimorfismo.

Definindo uma classe para um novo objeto contendo atributos e métodos.

```
class Contador:
    contador = 0 # atributo de classe

    def inc_maluco(self):
        self.contador = self.contador + 1
        return self.contador

    def inst(self):
        return 'Estou bem'

    @classmethod
    def inc(cls):
        cls.contador += 1
        return cls.contador

    @classmethod
    def dec(cls):
        cls.contador -= 1
        return cls.contador
```