APOSTILA BÁSICA PARA FUNDAMENTOS DO MINICURSO DE VISÃO COMPUTACIONAL

Autor: Caio M. Barros.

ÍNDICE

Sumário

COMENTÁRIOS

1 Prefácio

Antes de começar nosso minicurso, é importante ressaltar alguns pontos que irão ser retratados durante esse processo e também nossos focos e objetivos nesse curso.

Iremos abordar, principalmente, esses seguinte tópico:

- Python básico (instalação, booleanos, estruturas condicionais e lógicas, funções, repetições, listas, strings, dicionário, bibliotecas) - Que é o conteúdo dessa apostila.

Portanto, o mínimo requirido para começarmos esse curso é principalmente o interesse em aprender esses novos contéudos, como também conhecimentos básicos de lógica de programação, pois não será trabalhado a parte inicial do curso de Python/linguagem de programação básica em si.

Vale ressaltar também que como é um minicurso, muita coisa foi filtrada, então ficou mais o "core"do Python.

2 Entrada e Saída

Como já é visto em todas as linguagens de programação, segue um exemplo para apenas apresentar a identação de entrada e saída em python:

```
print ('Hello World') #pode-se usar aspas duplas ou aspas normais

Hello World
```

```
print("Hello", "World", sep = " || ") #funcao sep
print("Hello", "World", end = " || ") #funcao end

Hello || World
Hello World ||
```

```
>>> print('Bem vindo ao nosso minicurso')
nome = input('Digite seu nome: ')
print(nome)

>>> Caio
Caio
```

a função input() retorna sempre uma string. Então, caso quiséssemos pegar uma classe diferente de string, teremos que transformá-lo. Por exemplo:

```
>>> nome = input("Digite um numero: ")
int(nome) #vale para float, etc
print(nome)

12
```

3 Estruturas Básicas

3.1 Booleanos

Booleanos é um tipo de variável no Python que pode retornar somente dois valores, true ou false.

Para entender melhor essa parte, segue os exemplos em Python:

```
>>> print(1 == 1)

True
```

```
>>> print(1 != 1)

False
```

Basicamente, então, o boolean compara nossos valores e retorna aquilo seja verdade ou falso.

Note que estamos fazendo uma comparação entre dois valores, por isso a estrutura == está sendo usada. Caso queremos mudar o valor de alguma variável, ela será usada apenas com =

3.2 Estruturas Lógicas

Primeiramente, iremos abordar as estruturas que são diferentes em sua nomenclatura em diferentes linguagens de programação e/ou aquelas mais importantes. Iniciando-se pelas estruturas lógicas, temos:

and	Retorna True se as duas condições estão certas
or	Retorna True se pelo menos uma condição está certa
not	Inverte o resultado do booleano

Exemplos:

```
>>> x = 5
>>> y = 3
>>> print(x >= y and x == 5)

True
```

```
>>> x = 5
>>> y = 3
>>> print(x >= y and x != 5)

True
```

```
>>> x = 5
>>> y = 3
>>> print(not(x >= y and x == 5))

False
```

3.3 Estruturas de Identidade

Essas estruturas são utilizadas para comparar dados de uma certa variável.

Segue a tabela:

is	Retorna True se as duas variáveis são do mesmo objeto
is not	Retorna True se as duas variáveis NÃO são do mesmo objeto

Exemplos:

```
>>> x = 5
>>> print(type(x))
>>> print(type(x) is int)

<class 'int'>
True
```

```
>>> x = 5
>>> print(type(x) is float)

False
```

```
>>> x = 5
>>> print(type(x) is not int)

False
```

Como visto, a função type() retorna o tipo do dado objeto

3.4 Estruturas de Associação

in	Retorna True se o valor estiver contido no objeto comparado
not in	Retorna True se o valor não estiver contido no objeto comparado

Exemplos:

```
>>> print('C' in 'Caio')

True
```

```
>>> print('c' in 'Caio')

False
```

Note-se então que é case sensitivity.

```
>>> print('Ca' in 'Caio')

True
```

```
>>> print('CA' not in 'Caio')

False
```

3.5 Estruturas de Condição

Tem-se 3 condições, if, else e elif (else if)

- IF: Usa-se "if"para uma condição, se essa condição for satisfeita, então será satisfeita as linhas de códigos dessa condição. Por exemplo:

```
>>> x = 3
    y = 3

    if x == y:
     print ('Condicao Satisfeita')

Condicao Satisfeita
```

Caso não fosse verdade a condição do if, o terminal retornaria um valor vazio (None)

- IF-ELSE: Caso o código contenha esses dois Comandos, será lido primeiro a condição do if, caso seja falsa, retornará o Comando contido em "else"

```
>>> x = 3
    y = 5

    if x == y:
        print ('Condicao Satisfeita pelo if')

    else:
        print ('Condicao Satisfeita pelo else')

Condicao Satisfeita pelo else
```

- IF-ELIF: Mesmo raciocínio do IF-ELSE, porém terá um condição intermediária, chamada elif, que será lido depois do if e antes do else. Caso alguma condição do else-if seja satisfeita, será lida a primeira que for encontrada e o restante é descartada.

```
>>> print('Bem Vindo ao meu Banco, o que desejaria fazer: ')
print('1.Sacar dinheiro')
print('2.Depositar dinheiro')
print('3.Ver saldo')
print('4.Transferir dinheiro')

x = int(input('Escolha sua opcao: '))

if x == 1:
    print('Voce selecionou "Sacar dinheiro"')
elif x == 2:
    print('Voce selecionou "Depositar dinheiro"')
elif x == 3:
    print('Voce selecionou "Ver saldo"')
elif x == 4:
    print('Voce selecionou "Transferir dinheiro"')
else:
    print ('Opcao fora do intervalo proposto.')
```

Como visto, ele pede para o leitor digitar um numero desse intervalo, caso seja escolhido o número 2, por exemplo, será retornado "Voce selecionou "Depositar dinheiro"

LOOPS - FOR E WHILE

4 Estruturas de Repetição

4.1 O Comando for

4.1.1 Exemplos básicos

Exemplo básico 1:

```
>>> y = ['minicurso', 'python', 'engenharia', 'computacao', '2020']
for x in y:
    print(x)

<<< minicurso
python
engenharia
computacao
2020
```

Logo, usa-se uma variável que irá percorrer toda lista, tupla ou string.

Exemplo básico 2:

```
>>> y = 'Minicurso de Python feito pelo PET Engenharia de Computacao'
for x in y:
    x = x.lower()
    print(x, end='')
print('\n')
for x in y:
    x = x.upper()
    print(x, end='')

<<< minicurso de python feito pelo pet engenharia de computacao

MINICURSO DE PYTHON FEITO PELO PET ENGENHARIA DE COMPUTACAO
```

4.1.2 Função range()

```
>>> for i in range(2,3):
    for j in range(1,10):
        print('%d x %d = %d' % (i,j, i*j))

<<< 2 x 1 = 2
2 x 2 = 4
2 x 3 = 6
2 x 4 = 8
2 x 5 = 10
2 x 6 = 12
2 x 7 = 14
2 x 8 = 16
2 x 9 = 18
```

4.1.3 Função enumerate()

Transforma uma lista, que é da forma lista = ['item1', 'item2', ...] em uma tupla com duas iterações da forma (indice, itens). Portanto para quando queremos usar além dos itens, o índice também.

```
>>> fila = ['Ricardo',  'Jose', 'Jonathan', 'Felipe', 'Lara']
for posicao, nome in enumerate(fila): #funcao enumerate transforma a fila em uma tupla(indice, item)
    print('posicao: %s' % (posicao+1))
    print('nome: %s' % nome)
    print('---')

<<< posicao: 1
nome: Ricardo
---
posicao: 2
nome: Jose
---
posicao: 3
nome: Jonathan
---
posicao: 4
nome: Felipe
---
posicao: 5
nome: Lara
---
```

4.2 O Comando while

4.2.1 Introdução

Basicamente usa-se while para executar uma função até que a condição imposta seja atingida. Segue os exemplos:

```
>>> lista = ['a','b','c','d','e']
tamanho = len(lista)
i = 0
while i < tamanho: #condicao imposta
        print(lista[i], end='|')
        i = i + 1 #printara o primeiro caso, "a", depois incrementa +1, gerando i = 1 toda vez no loop.

<<< a|b|c|d|e|
```

LISTAS

Basicamente, iremos usar exemplos para explicar suas funções e sua utilização.

5 Listas/Lists

5.1 Introdução

```
>>> minicurso = ['aulas', 10, 'Python', 2342, 'futebol']
print(minicurso[1])
print(minicurso[2])
print(minicurso[3])
print(minicurso[-1])    #'-1' representa o ultimo valor da lista
print(minicurso[1:4])   #intervalo de uma lista
print(minicurso[:-1])   #":" separa a lista ate determinado valor imposto

10
```

```
Python
2342
futebol
[10, 'Python', 2342]
['aulas', 10, 'Python', 2342]
```

5.2 Função: Modificar o valor de uma lista:

Para mudar um valor da lista, basta inferirmos isso:

```
>>> minicurso = ['aulas', 10, 'Python', 2342, 'futebol']
minicurso[3] = 'novaPalavra'
print(minicurso)

['aulas', 10, 'Python', 'novaPalavra', 'futebol']
```

para mudar valores em quantidade, precisa-se:

```
>>> minicurso = ['aulas', 10, 'Python', 2342, 'futebol']
minicurso[:3] = ['palavra1', 'palavra2', 'palavra3']
print(minicurso)

['palavra1', 'palavra2', 'palavra3', 2342, 'futebol']
```

5.3 Função: Adição de valores em uma lista

5.3.1 Comando append

Augmenta um valor depois do último da lista

```
>>> minicurso = []
print(minicurso)
minicurso.append('primeiraPalavra')
print(minicurso)
minicurso.append('segundaPalavra')
print(minicurso)

[]
['primeiraPalavra']
['primeiraPalavra', 'segundaPalavra']
```

5.3.2 Comando insert

Augmenta em um determinado índice da lista

```
>>> minicurso = [1,2,3,4,5,6]
minicurso.insert(2,'palavraInserida')
print(minicurso)

[1, 2, 'palavraInserida', 3, 4, 5, 6]
```

5.3.3 Comando extend

Concantena duas listas

```
>>> minicurso = [1,2,3]
listaconc = ['p1','p2']
minicurso.extend(listaconc)
print(minicurso)

[1, 2, 3, 'p1', 'p2']
```

5.4 Função: Remoção de valores em uma lista

5.4.1 Comando remove

Retira o primeiro caso que aparecer na lista

```
>>> minicurso = [1,2,1,2,5,2,3,4]
minicurso.remove(2)
print(minicurso)

[1, 1, 2, 5, 2, 3, 4]
```

5.4.2 Comando pop

Retira o ultimo elemento da lista

```
>>> minicurso = [1,2,1,2,5,2,3,4]
minicurso.pop()
print(minicurso)

[1, 2, 1, 2, 5, 2, 3]
```

5.5 Demais Funções

5.5.1 Comando len

Indica a quantidade de elementos na lista

```
>>> minicurso = [1,2,1,2,5,2,3,4]
print(len(minicurso))

8
```

5.5.2 Comando sum

Soma os valores dos elementos

```
>>> minicurso = [1,2,1,2,5,2,3,4]
print(sum(minicurso))

20
```

5.5.3 Comando max

Indica o maior elemento da lista

```
>>> minicurso = [1,2,1,2,5,2,3,4]
print(min(minicurso))
print(max(minicurso))

1
5
```

5.5.4 Comando sorted

Organiza a lista em ordem alfabética

```
>>> minicurso = ['j','k','w','m','a','b']
print(sorted(minicurso))

['a', 'b', 'j', 'k', 'm', 'w']
```

5.5.5 Comando sort

Organiza os numeros em ordem crescente

```
>>> minicurso = [5,3,10,14,2]
minicurso.sort()
print(minicurso)

[2, 3, 5, 10, 14]
```

5.5.6 Comando reverse

Inverte a lista

```
>>> minicurso = [5,3,10,14,2]
minicurso.reverse()
print(minicurso)

[2, 14, 10, 3, 5]
```

6 Tuplas/Tuples

6.1 Introduçao à Tupla

Basicamente é uma lista, porém é utilizada para dar um range de certos valores numéricos. Vale ressaltar também que, diferente da lista, ela não pode ser alterada após a declaração de uma tupla.

Para declarar uma tupla, utiliza-se () ao invés de []

Segue um exemplo:

```
>>> minicurso = (1,3,10,400)
minicurso.append(20)

Traceback (most recent call last):
  File "c:/Users/Caio/Desktop/minicurso_python.py", line 2, in <module>
    minicurso.append(20)
AttributeError: 'tuple' object has no attribute 'append'
```

Mais usual para:

```
>>> 1.5.as_integer_ratio()
(3, 2) #tupla
```

7 Dicionário/Dictionary

7.1 Introduçao ao Dicionário

Será construido com o uso de chaves ""e será composto pelo índice e pelo valor atribuído a esse índice, podendo ser uma string, ou qualquer outra coisa.

Exemplos:

```
>>> carros = {1: 'Opala', 2: 'Gol', 3: 'Hilux'}
for carro in carros:
    print(carros[carro], end=' ')
print('\n-------')
print(type(carros))
print('-------')
print(carros)

<<< Opala Gol Hilux
-------
<class 'dict'>
-------
{1: 'Opala', 2: 'Gol', 3: 'Hilux'}
```

7.2 Comando items()

Podemos usar um comando similar ao enumerate() em lista, que é o comando items()

```
>>> carros = {1: 'Opala', 2: 'Gol', 3: 'Hilux'}
for indice, nome in carros.items():
    print('index = %d' % indice)
    print('valor correspondente ao index = %s' % nome)
    print('~~~~~~')

<<< index = 1
valor correspondente ao index = Opala
~~~~~~
index = 2
valor correspondente ao index = Gol
~~~~~~
index = 3
valor correspondente ao index = Hilux
~~~~~~
```

7.3 Comando get()

Basicamente usa-se esse comando quando queremos indicar um valor que não está no dicionário anteriormente, Veja:

```
>>> carros = {1: 'Opala', 2: 'Gol', 3: 'Hilux'}
print(carros.get(2))
print(carros.get(4))
print(carros.get(4, 'Brasilia'))

<<< Gol
None
Brasilia
```

7.4 Adicionando elementos ao dicionário

Declara o dicionário e augmenta-se os valores em cada índice que queira

```
>>>carros = {}
carros[1] = 'Opala'
carros[2] = 'Gol'
carros[3] = 'Hilux'
print(carros)

<<< {1: 'Opala', 2: 'Gol', 3: 'Hilux'}
```

Para mudar um pouco o valor dos índices, tome um exemplo usando string ao invés de int

```
>>> carros = {}
carros['carro1'] = 'Opala'
carros['carro2'] = 'Gol'
carros['carro3'] = 'Hilux'
print(carros)

<<< {'carro1': 'Opala', 'carro2': 'Gol', 'carro3': 'Hilux'}
```

7.5 Comando del - Retira elementos do dicionário

```
>>> carros = {'carro1': 'Opala', 'carro2': 'Gol', 'carro3': 'Hilux'}
del carros['carro2']
print(carros)

<<< {'carro1': 'Opala', 'carro3': 'Hilux'}
```

8 Conjuntos/Set

8.1 Introduçao aos Conjuntos

Os conjuntos, ou set, como é descrito em Python, diferem dos dicionário e das listas porque eles são declaradamente sem índice e sem ordem. Ele é parecido com o dicionário, mas não precisa colocar o índice que fica antes de cada valor atribuído e se difere com as listas pois quando é printado o set, ele printa os valores sem ordem alguma e difere de uma tupla porque no set pode-se adicionar. Veja os seguintes exemplos:

```
>>> f = {'Abacaxi', 'Melao', 'Uva', 'Morango'}
print(f)

<<< {'Melao', 'Morango', 'Abacaxi', 'Uva'}
```

Se rodarmos mais uma vez o mesmo código, será printado, provavelmente, um novo output diferente do primeiro

```
>>> f = {'Abacaxi', 'Melao', 'Uva', 'Morango'}
print(f)

<<< {'Morango', 'Melao', 'Uva', 'Abacaxi'}
```

Pode-se usar um loop para printar cada elemento do set

```
>>> f = {'Abacaxi', 'Melao', 'Uva', 'Morango'}
for i in f:
    print(i, end=' ')

<<< Abacaxi Uva Melao Morango
```

8.2 Comando add()

Adicionando itens em um set

```
>>> f = {'Abacaxi', 'Melao', 'Uva', 'Morango'}
f.add('Caja')
print(f)

<<< {'Morango', 'Melao', 'Abacaxi', 'Caja', 'Uva'}
```

8.3 Comando update()

Adição de vários itens ao mesmo tempo

```
>>> f = {'Abacaxi', 'Melao', 'Uva', 'Morango'}
f.update(['Caja', 'Abacate', 'Limao', 'Banana'])
print(f)

<<< {'Limao', 'Uva', 'Abacate', 'Melao', 'Morango', 'Abacaxi', 'Caja', 'Banana'}
```

8.4 Demais funções em Conjuntos

Method	Description
add()	Adds an element to the set
clear()	Removes all the elements from the set
copy()	Returns a copy of the set
difference()	Returns a set containing the difference between two or more sets
difference_update()	Removes the items in this set that are also included in another, specified set
discard()	Remove the specified item
intersection()	Returns a set, that is the intersection of two other sets
intersection_update()	Removes the items in this set that are not present in other, specified set(s)
isdisjoint()	Returns whether two sets have a intersection or not
issubset()	Returns whether another set contains this set or not
issuperset()	Returns whether this set contains another set or not
pop()	Removes an element from the set
remove()	Removes the specified element
symmetric_difference()	Returns a set with the symmetric differences of two sets
symmetric_difference_update()	inserts the symmetric differences from this set and another
union()	Return a set containing the union of sets
update()	Update the set with the union of this set and others

Figura 2: Fonte: https://www.w3schools.com/python/python_sets.asp

FUNÇÕES EM PYTHON

Funções é um dos comandos mais importantes, versáteis e úteis para um programador. Essa parte será justamente para descrevermos e explicarmos tal fato.

9 Funções em Python

9.1 Criando uma Função

```
>>> def funcaoDoSegundoGrau(x):
    return x^2 + x + 1

print(funcaoDoSegundoGrau(2))

<<< 7
```

Sempre será esse tipo de estrutura. Usando o comando def.

9.2 Funções de Ajuda

9.3 Definindo Funções

9.4 Funções com Docstring

É basicamente para podermos usar o comando help(NomeDaFuncao)

Com ela, podemos citar um comentário dentro da nossa função e caso precise, o usuário/programador pode analisar o comentário usando a função help.

```
>>> def funcaoDoSegundoGrau(x):

    """basta chamar a funcao com um valor desejavel,
    que ira substituir e retornar o valor automaticamente"""
    return x^2 + x + 1

help(funcaoDoSegundoGrau)
```

```
print(funcaoDoSegundoGrau(2))

Help on function funcaoDoSegundoGrau in module __main__:

funcaoDoSegundoGrau(x)
    basta chamar a funcao com um valor desejavel,
    que ira substituir e retornar o valor automaticamente

7
```

9.5 Funções sem Retorno

Basicamente conseguimos criar funções sem retorno, mas caso seja necessário chamar/retornar um valor pois a função pede, será retornado o valor "None"que é o "null"para outras linguagens.

```
>>> def funcaoDoSegundoGrau(x):
    x^2 + x + 1

print(funcaoDoSegundoGrau(2))

<<< None
```

9.6 Funções com Argumento padrão

É possivel criar uma função que um dos argumentos dela já seja uma definida. Segue o exemplo:

```
def calculadora_salario(valor, horas=2.55):
    return horas * valor

valor_total = calculadora_salario(14)

print(valor_total)

<<< 35.699999999999996
```

Perceba que horas=2.55 é um argumento padrão.

BIBLIOTECAS

Será abordado algumas aplicações, funcionalidades e objetivos para os quais se estuda Python, dentre eles destacam-se algumas bibliotecas e APIs que estão no foco dos mais variados temas, das quais se destacam: Machine Learning, Deep Learning, Image Processing e Data Science.

Será destacado as referências de maior relevâncias dentre as quais foram usadas pelo autor.

10 Referências

- kaggle: https://www.kaggle.com/learn/python
- udemy: https://www.udemy.com/course/curso-de-programacao-em-python-do-basico-ao-avancado/
- hackerearth: https://www.hackerearth.com/pt-br/practice/python/
- w3schools: https://www.w3schools.com/python/
- geekforgeeks: https://www.geeksforgeeks.org