Luiz Felipe Nagatani rabello

Relatório LAMIA - Módulo 2

No primeiro vídeo é apresentado a parte mais básica do python como a instalação de um interpretador e das extensões para utilizar python no VScode, assim como a função do shebang #!python3 que declara o interpretador a ser utilizado.

Logo em seguida são apresentados os packages que funcionam como as bibliotecas do python, na imagem abaixo são expostos alguns packages declarados.

```
import HandTrackingModule
import tensorflow
import cv2
```

.

Foi mostrado como comentar o código usando o “#” e como declarar variáveis, utilizando aspas para declarar strings e utilizando fstrings para interpretar variáveis dentro de strings. Outra função exposta foi a “multiplicação” de strings utilizando o “*” conforme o exemplo abaixo.

```
texto = 'talalalala '
idade = 22
print(2 * texto)
```

```
[Running] python -u "c:\Users\shink\LAMIA 1.py"
talalalala talalalala
```

para atribuir o valor de uma variável utilizando a entrada do usuário deve se usar o comando input, que atribuirá o valor informado pelo usuário à variavel, pode-se usar uma string como parâmetro para o comando, afim de expor texto para o usuário antes da entrada.

```
altura = float(input('qual sua altura? '))
print(f'sua altura é {altura}m')
```

É possível manipular listas/arrays, utilizando append() para adicionar um elemento novo na lista, insert() para inserir um elemento numa determinada posição, diferente dos conjuntos, que por serem indexados não podem ter seus elementos manipulados individualmente.

```
nums = [1, 2, 3]
print(type(nums))
nums.append(int(input()))
print(nums)
```

listas

```
conjunto = {2,4,6,8,10,12}
print(conjunto)
```

conjunto

Também existem as Tuplas, que estão para as strings como as listas estão para as variáveis numerais

```
nomes = ('luiz ', 'felipe ', 'nagatani ', 'rabello ')
print(nomes)
```

Outro tipo existente é o dicionário, que não pode ter dados duplicados e é exposto em chave/valor.

```
aluno ={
    'nome': 'Luiz Felipe Nagatani Rabello',
    'nota': 9.0,
    'ativo': True
}

print(aluno['nome'])
print(aluno['nota'])
print(aluno['ativo'])
```

Em seguida nós tempos os operadores, sendo eles unários, binários e ternários

- Unários são os operadores que dependem de um só operando, como por exemplo os operadores de negação.

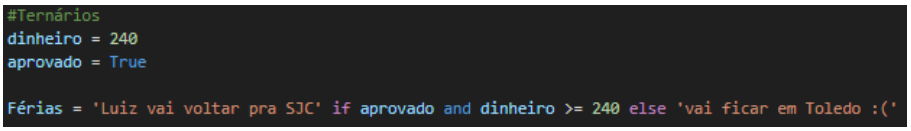
```
#unário
x =2
print(True)
print(not True)
print(-x)
```

- Binários são os operadores que dependem de dois operandos, são divididos em aritméticos(adição, subtração, divisão, módulo e multiplicação) , relacionais (!=, ==, <=, <, etc.) atribuição(+=, -=, *=)

```
y = 2
z = 3
#aritméticos
print(y + z)
print(y - z)
print(y / z)
print(y % z)
print(y * z)
#relacionais
print(y != z)
print(y < z)
print(y == z)
```

```
#atribuição| atribui um novo valor a variavel após realizar a operação na frente do "="
final = 3
final += 3
final -= 4
final /= 5
final *= 8
final %= 5
print(final)
#lógicos
a1 = True
a2 = False
a3 = True
print (a1 and a2 and a3)
print (a1 or a2 or a3)
```

- Ternários avaliam condições porém não executam nada, ao contrário do if e do else, exemplos na imagem abaixo

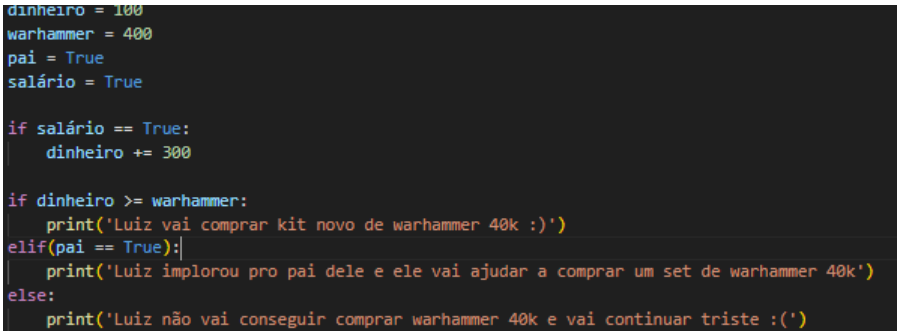


```
#Ternários
dinheiro = 240
aprovado = True

Férias = 'Luiz vai voltar pra SJC' if aprovado and dinheiro >= 240 else 'vai ficar em Toledo :('
```

No python um bloco de código é formado utilizando indentação, diferente de outras linguagens, como por exemplo o C, que são utilizadas as chaves

If e else são estruturas condicionais feitas para avaliar condições e executar comandos caso a condição for/não for cumprida, com if retornando true se a condição for cumprida, else retornando true caso if não for cumprido e elif caso if não for cumprido mas outra condição for cumprida, exemplo:

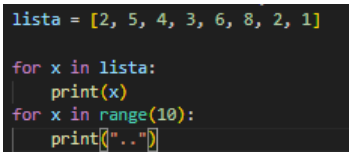


```
dinheiro = 100
warhammer = 400
pai = True
salário = True

if salário == True:
    dinheiro += 300

if dinheiro >= warhammer:
    print('Luiz vai comprar kit novo de warhammer 40k :)')
elif(pai == True):
    print('Luiz implorou pro pai dele e ele vai ajudar a comprar um set de warhammer 40k')
else:
    print('Luiz não vai conseguir comprar warhammer 40k e vai continuar triste :(')
```

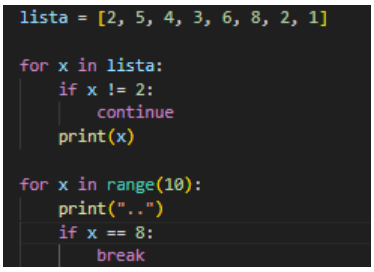
For é um comando utilizado como estrutura de controle para realizar iteração,e pode ser utilizado tanto pra iterar dentro de listas, tuplas, strings e sets, quanto pra iterar com um valor como ponto de parada.



```
lista = [2, 5, 4, 3, 6, 8, 2, 1]

for x in lista:
    print(x)
for x in range(10):
    print("..")
```

Há formas de manipular a iteração, utilizando pass, break e continue. Pass irá permitir voce criar um for vazio sem que seja marcado um erro, continue irá continuar a iteração caso uma condição declarada anteriormente seja cumprida e break ira encerrar o for.

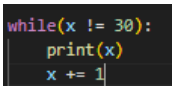


```
lista = [2, 5, 4, 3, 6, 8, 2, 1]

for x in lista:
    if x != 2:
        continue
    print(x)

for x in range(10):
    print("..")
    if x == 8:
        break
```

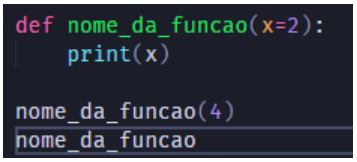
Outra estrutura de controle é o while, que executa um comando enquanto uma condição for cumprida, avaliando se a condição ainda é cumprida toda vez que surgir uma nova iteração



```
while(x != 30):
    print(x)
    x += 1
```

No segundo vídeo são abordadas as funções, que são declaradas utilizando a seguinte estrutura: “def nome_da_função(parâmetro):”, ao utilizar uma função importada de um módulo, deve se usar o namespace do módulo, isso é requisitado no python para evitar conflitos e sobrescrita entre funções.

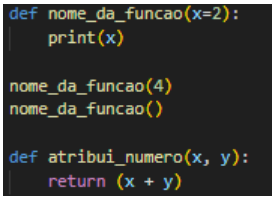
É possível criar uma função com um parâmetro base porém alterável, utilizando a estrutura “def nome_da_funçao(parâmetro = x):” neste caso sempre que a função for executada como nome_da_funçao() o valor do parâmetro será x, porém caso ela seja executada como nome_da_funçao(20) por exemplo, o valor do parâmetro será 20.



```
def nome_da_funcao(x=2):
    print(x)

nome_da_funcao(4)
nome_da_funcao
```

uma função pode ser utilizada como parâmetro para outro bloco, no caso deve-se colocar um return na função, permitindo que ela retorne um dado que será utilizado em outro ponto no código



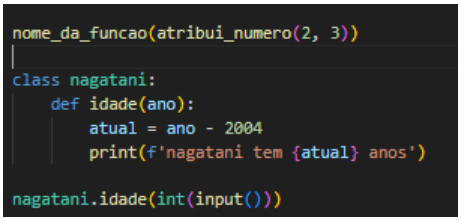
```
def nome_da_funcao(x=2):
    print(x)

nome_da_funcao(4)
nome_da_funcao()

def atribui_numero(x, y):
    return (x + y)
```

Para criar uma estrutura que se assemelha a um dicionário, porém que inclui funções, pode ser utilizada as classes, usadas para criação de código modular

Por último, é explicado sobre os decorators, que modificam o comportamento de funções sem modificar seu código, alguns exemplos são o @classmethod que transforma um método em uma classe sem precisar instanciar uma classe, e o @staticmethod que faz o método se comportar com uma função normal mesmo pertencendo a uma classe



```
nome_da_funcao(atribui_numero(2, 3))

class nagatani:
    def idade(ano):
        atual = ano - 2004
        print(f'nagatani tem {atual} anos')

nagatani.idade(int(input()))
```