Resumo do minicurso

- UNIDADE 1 – Introdução ao Python:
  1. História
  2. Aplicações
  3. CompilaçãoIDE
  4. Por que Python?
  5. Vantagens
  6. Sintaxe da Linguagem
- UNIDADE 2 – Download e Instalação:
  1. Download e Instalação do interpretador
  2. Download e Instalação do Pycharm
- UNIDADE 3 – Conceitos Básicos:
  1. Hello World
  2. Variáveis
  3. Expressões Aritméticas
  4. Expressões Lógicas
  5. Entrada e Saída de dados
  6. Atribuição
  7. Bibliotecas
- UNIDADE 4 – Desvios Condicionais:
  1. If, Elif e Else
- UNIDADE 5 – Laços de Repetição:
  1. For
  2. While
- UNIDADE 6 – Strings e Conjuntos de Dados:
  1. Strings
  2. Listas, Tuplas e Dicionários
  3. Matrizes
- UNIDADE 7 – Funções:
  1. Funções
- UNIDADE 8 – Programação Orientada a Objetos:
  1. Abstração
  2. Encapsulamento
  3. Herança
  4. Polimorfismo

Python 101

UNIDADE 6 – Strings e Conjuntos de Dados:

1. Strings
2. Listas, Tuplas e Dicionários
3. Matrizes

UNIDADE 6

Strings:

Na unidade 3.5 foram mencionadas algumas informações sobre strings em Python.

Nesta seção, esses **‘builtins’** (funções nativas presentes no interpretador sem precisarem ser importadas) são abordados de forma mais ampliada.

Segundo Borges (2010), as Strings são imutáveis, ou seja, não é possível adicionar, remover ou substituir seus caracteres sem que outra String seja criada.

Downey (2015) conceitua uma String como uma sequência, como uma coleção ordenada de caracteres.

UNIDADE 6

Como visto anteriormente, a inicialização de uma String pode ser tanto com aspas simples, como com aspas duplas (ou normais). E, como Strings são sequências fixas de caracteres, cada caractere está alocado na memória ocupando uma posição.

Dessa forma, é possível acessar cada caractere de uma String, acessando sua respectiva posição através da utilização de colchetes “[]”. O código abaixo ilustra o que foi dito.

```
>>>    = "Python top!"
>>>   [0]
‘P'
>>>   [0 : 5]
'Python’
>>>   [-4 : -1]
‘top’
```

UNIDADE 6

Como uma String é uma sequência, então essa pode ser percorrida por laços de repetição (for ou while). Abaixo é ilustrado o que foi dito utilizando os laços “for” e “while”.

```
## Utilizando o laço while:
s = "Python top!"
i = 0
while i != len(s):
        i += 1
        print(s[-i])
```

```
## Utilizando o laço for:
s = "Python top!"
for j in range(len(s)):
        print(s[-j-1])
```

UNIDADE 6

Além disso, Strings podem ser concatenadas utilizando o operador “+” ou podem ser interpoladas utilizando o operador “%” junto da letra que corresponde ao tipo da variável considerada.

Por fim, destaca-se que Strings possuem métodos próprios que são muito úteis em inúmeros exemplos. Alguns dos mais utilizados são:

- **upper()** → eleva todos os caracteres da String para maiúsculos;
- **find()** → busca um determinado caractere dentro da String e retorna sua posição;
- **count()** → conta o número de repetições de um caractere dentro da String e
- **split()** → ‘recorta’ a String considerada, transformando-a numa lista.

Outros métodos podem ser facilmente encontrados em
https://docs.python.org/2/library/string.html

UNIDADE 6

```
>>> s = 'Apostila de python top!'
>>> print('\nA string "' + s + '" em caixa alta , é: ', s.upper())
A string "apostila de python top!" em caixa alta , é: APOSTILA DE
PYTHON TOP!
>>>
>>> print('\nA posição do caractere "!", é: ', s.find('!'))
A posição do caractere "!", é: 22
```

UNIDADE 6

```
>>> print('\nO número de repetições do caractere "i", é: ',
s.count('i'))
O número de repetições do caractere "i", é: 1
>>>
>>> print('\nA string recortada em todas as letras "o", é: ',
s.slpit('o'))
A string recortada em todas as letras "o", é: ['ap', 'stila de pyth',
'n t', 'p!']
```

Exercício de Revisão: Unidade 6.1

Faça um programa que leia uma String e a imprima com todas os caracteres em MAIÚSCULO.

- Utilize o método de string ‘upper()’

EXERCÍCIO EXEMPLO.

Exercício de Revisão: Unidade 6

```
a = input('Digite a string: ')
print('A string é: ', a.upper())
```

Exercício de Revisão: Unidade 6.1

Fazer um programa para ler uma string e contar quantas vezes um determinado caractere aparece na string. O caractere deverá ser informado pelo usuário.

- Não Utilize o método de string ‘count()’

Exercício de Revisão: Unidade 6.1

```
string = input('Digite a string: ')
letra = input('Digite a letra procurada: ')
count = 0
for i in string:
    if i == letra:
        count += 1
print('O número de repetições da letra "',
      letra, '" é: ', count)
```

Exercício de Revisão: Unidade 6.1

Fazer um programa para ler uma string e contar quantas vezes um determinado caractere aparece na string. O caractere deverá ser informado pelo usuário.

- Utilize o método de string ‘count()’

Exercício de Revisão: Unidade 6.1

```
string = input('Digite a string: ')
letra = input('Digite a letra procurada: ')
print('O númeor de repetições é: ', string.count(letra))
```

UNIDADE 6

Listas:

Em Python, uma lista é uma sequência mutável de n valores que podem ser de qualquer tipo (inclusive outras Listas, tupas e dicionários).

De forma simples, uma lista pode ser entendida como um vetor de elementos que podem ser de qualquer tipo. As Listas são exatamente iguais às Strings, exceto pelo fato de Strings serem imutáveis e Listas serem mutáveis.

```
>>> lista = ['Pode conter qualquer tipo de valor',0 , 5 + 8j, ['Outras listas', 'por exemplo'], 5.5]
>>> lista
['Pode conter qualquer tipo de valor',0 , (5+8j), ['Outras listas', 'por exemplo'], 5.5]
```

UNIDADE 6

Listas:

As listas podem ser percorridas, “fatiadas” e concatenadas da mesma forma que as Strings. A diferença é que em se tratando de Strings, cada elemento é um caractere e, em se tratando de listas, cada elemento pode ser qualquer tipo de variável.

Além disso, uma String pode ser convertida para uma lista (como já foi visto) e uma lista pode ser convertida para uma string. Abaixo são ilustradas estas duas conversões.

```
>>> a = 'Apostila de Python é top!'
>>> # Converter cada letra: list()
. . .
>>> b = list(a)
>>> b
['A', 'p', 'o', 's', 't', 'i', 'l', 'a', ' ', 'd', 'e', ' ', 'P', 'y’, 't', 'h', 'o', 'n', 'é', ' ', 't’, 'o', 'p', '!']
```

UNIDADE 6

Listas:

```
>>> # Converter por espaços ou por elementos previamente indicados:
split()
. . .
>>> b = a.split()
>>> b
['Apostila', 'de', 'Python', 'é', 'top!']
```

UNIDADE 6

Listas:

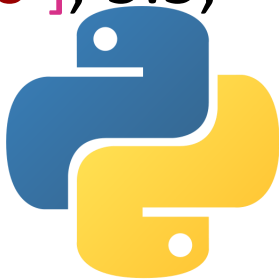
```
>>> # Converter lista para string: join()
. . .
>>> # join() concatena todos os elementos da lista. Caso a lista não possua espaços posicionados de forma correta , então é necessário que o join() seja utilizado a partir de um ' ' (espaço) ou de uma variável com a mesma finalidade.
. . .
>>> a = ' '.join(b)
>>> a
'Apostila de Pyhton é top!'
```

UNIDADE 6

Listas:

Listas em Python possuem alguns métodos e algumas operações que auxiliam e muito o trabalho do programador. A seguir serão mostrados alguns exemplos de utilização desses métodos e dessas operações. Para se obter mais informações sobre outros métodos e sobre outras operações, sugerimos Borges (2010), Downey (2015) e a documentação do Python em: https://docs.python.org/2/tutorial/datastructures.html

```
>>> lista = ['Pode conter qualquer tipo de valor',0, 5 + 8j, ['Outras listas', 'por exemplo'], 5.5]
>>> # Para acrescentar um novo elemento após o último elemento:
. . .
>>> lista.append('Novo elemento')
>>> lista
['Pode conter qualquer tipo de valor',0 , (5+8j), ['Outras listas', 'por exemplo'], 5.5, 'Novo elemento']
```



UNIDADE 6

Listas:

```
>>> # Para excluir um elemento "n" da lista:
. . .
>>> lista.remove(5.5)
>>> lista
['Pode conter qualquer tipo de valor', 0, (5+8j), ['Outras listas', 'por exemplo'], 'Novo elemento']
```

UNIDADE 6

Listas:

```
>>> # Para ordenar todos os elementos da lista em ordem ascendente: sort() obs.: todos os elementos devem ser do mesmo tipo
. . .
>>> lista = [-1, 50, 9, 7, 0, -6, 100]
>>> lista.sort()
>>> lista
[-6, -1, 0, 7, 9, 50, 100]
```

UNIDADE 6

Listas:

```
>>> # Para inverter todos os elementos da lista: reverse()
. . .
>>> lista = ['Pode conter qualquer tipo de valor', 0, 5 + 8j, ['Outras listas', 'por exemplo'], 'Novo elemento']
>>> lista.reverse()
>>> lista
[5.5, ['Outras listas', 'por exemplo'], (5+8j), 0, 'Pode conter qualquer tipo de valor']
```

UNIDADE 6

Listas:

```
>>> # Para retirar o último elemento da lista: pop() obs.: este método retorna o elemento retirado.
. . .
>>> print('O elemento retirado da lista , é: ', lista.pop())
O elemento retirado da lista , é: Pode conter qualquer tipo de valor
>>> lista
[5.5, ['Outras listas', 'por exemplo'], (5+8j), 0]
```

Obs.: **“sort()”** e **“pop()”** são muito utilizados aos se trabalhar com filas e pilhas. → Modelagem.

Exercício de Revisão: Unidade 6.2

Crie um programa que leia 15 elementos de um vetor A. Construir um vetor B de mesmo tipo, observando a seguinte lei de formação: Todo elemento de B deve ser o quadrado do elemento de A correspondente. Apresentar os 2 vetores no final do algoritmo.

Exercício de Revisão: Unidade 6.2

```
A = []
B = []
for i in range(15):
    x = int(input('Digite o número: '))
    A.append(x)
    B.append(x**2)
print('\nLista A: ', A, '\nLista B:', B)
```

Exercício de Revisão: Unidade 6.2

Escrever um programa que leia 5 números reais e imprima-os na ordem inversa.

Exercício de Revisão: Unidade 6.2

```
num = []
for i in range(5):
    num.append(float(input('N°: ')))
for i in range(len(num)):
    print(num[-i-1])
```

Exercício de Revisão: Unidade 6.2

Crie um vetor de 100 elementos em que cada dado é a letra ‘P’ ou ‘I’, dependendo da sua posição ser par ou ímpar. Feito isso, apresente o resultado na tela.

Exercício de Revisão: Unidade 6.2

```
vetor = []
for i in range(1,100,1):
    if i % 2 == 0:
        vetor.append('P')
    else:
        vetor.append('I')
print(vetor)
```

Exercício de Revisão: Unidade 6.2

Crie um programa que leia 5 números inteiros e os armazene em uma lista de tal forma que todos os números maiores ou iguais que o primeiro fiquem ao lado direito e todos os menores fiquem ao lado esquerdo.

EXERCÍCIO EXEMPLO.

Exercício de Revisão: Unidade 6.2

```
num = []
for i in range(5):
    n = int(input('N°: '))
    if len(num) == 0:
        num.append(n)
        x = n  # N° a ser comparado
    else:
        if n >= x:
            num.append(n)
        elif n < x:
            num.insert(0, n)
    # print(num)
print(num)
```

Exercício de Revisão: Unidade 6.2

Faça um programa que coloque em ordem 5 números quaisquer de uma lista.

- Obs.: Utilizar o método “sort()”.

Exercício de Revisão: Unidade 6.2

```
A = [2, 1, -9, 0, 7]
A.sort()
print(A)
```

Exercício de Revisão: Unidade 6.2

Faça um programa que remova todos os números pares de uma lista de 0 até 10.

- Obs.: Utilizar o método “remove()”.

Exercício de Revisão: Unidade 6.2

```
lista = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
for i in lista:
    if i % 2 == 0:
        lista.remove(i)
print(lista)
```

UNIDADE 6

Tuplas:

As Tuplas são muito semelhantes às Listas, exceto pelo fato de que as Tuplas são imutáveis. Ou seja, as operações mostradas anteriormente em que foi possível mudar cada elemento da lista "lista[0] = qualquer coisa", em Tuplas não são possíveis.

De forma concisa, Tuplas não permitem apagar, acrescentar e nem realizar atribuições a Tuplas já criadas. Abaixo é mostrado a sintaxe de uma Tupla.

```
>>> tupla = (0, 1, 2, 'muito parecio com listas')
>>> tupla
(0, 1, 2, 'muito parecio com listas')
>>> tupla = 0, 1, 2, 'muito parecio com listas'
>>> tupla
(0, 1, 2, 'muito parecio com listas')
>>> # Parenteses são opcionais.
```

UNIDADE 6

Tuplas:

Para criar-se uma Tupla com apenas um elemento, deve-se escrever: “nome_da_variável = (elemento)”. É necessário se escrever a vírgula pois caso contrário o interpretador irá entender que a variável “nome_da_variável” é apenas uma variável do mesmo tipo do “elemento”.

Além disso, é possível se converter listas em tuplas. Para isso, mostra-se o exemplo abaixo.

UNIDADE 6

Tuplas:

Além disso, é possível se converter listas em tuplas. Para isso, mostra-se o exemplo abaixo.

```
>>> # Lista para Tupla:
. . .
>>> lista = [0, 1, 2, 3]
>>> tupla = tuple(lista)
>>> tupla
(0, 1, 2, 3)
```

```
>>> # Tupla para Lista:
. . .
>>> lista = list(tupla)
>>> lista
[0, 1, 2, 3]
```

UNIDADE 6

Tuplas:

Um exemplo de utilização de uma tupla pode ser para se separar endereços de log-in dos seus respectivos domínios (Hotmail, outlook, etc). Abaixo é ilustrada essa operação.

```
>>> email = 'apostila@muitoTop.com'
>>> u_name , dominio = email.split("@")
>>> print('\nNome do usuário: ', u_name , '\nNome do domínio: ', dominio)
```

```
Nome do usuário: minicurso
Nome do domínio: muitoTop.com
```

UNIDADE 6

Tuplas:

Embora uma Tupla seja imutável, essa pode ser formada por elementos mutáveis (como listas) e então esses elementos podem ser atualizados, desde que se respeite o formado da Tupla. O exemplo presente abaixo ilustra esse caso.

UNIDADE 6

Tuplas:

```
>>> tupla = [0, 1], 'a', 'b'
>>> # Neste caso , o primeiro elemento da tupla é uma lista (mutável). Então para se modificar esse elemento , deve-se acessar a posição desse elemento na tupla e então utilizar os métodos da lista. Não se pode atribuir uma nova lista ao elemento da Tupla.
. . .
>>> tupla[0].append(2)
>>> tupla
([0, 1, 2], 'a', 'b')
>>>
>>> # Jeito errado:
. . .
>>> tupla[0] = [0, 1, 2]
Traceback (most recent call last):
        File "<stdin >", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

Exercício de Revisão: Unidade 6.3

Faça um programa que crie uma tupla com o que você quiser dentro e depois altere o último elemento para o número 3.

Obs.: Tuplas são imutáveis.

Exercício de Revisão: Unidade 6.3

```
tupla = (4, 5, 7)
print(tupla)
tupla = list(tupla)
#  Converter a tupla para
#        lista e então alterar.
tupla[-1]=3
print(tuple(tupla))
```

UNIDADE 6

Dicionários:

Segundo Borges (2010), um dicionário é uma lista de associações composta por uma chave (de tipo imutável) e estruturas correspondentes às chaves que podem ser mutáveis ou não. Isso caracteriza os Dicionários como sendo mutáveis (pois os valores em si são mutáveis, exceto as chaves).

Segundo Downey (2015), os dicionários são um dos melhores recursos do Python.

UNIDADE 6

Dicionários:

```
>>> # Forma geral:

. . .
>>> dicionario = {'chave1' : 1, 'chave2' : 2, 'chave3' : 3}
>>> dicionario
{'chave1' : 1, 'chave2' : 2, 'chave3' : 3}
```

UNIDADE 6

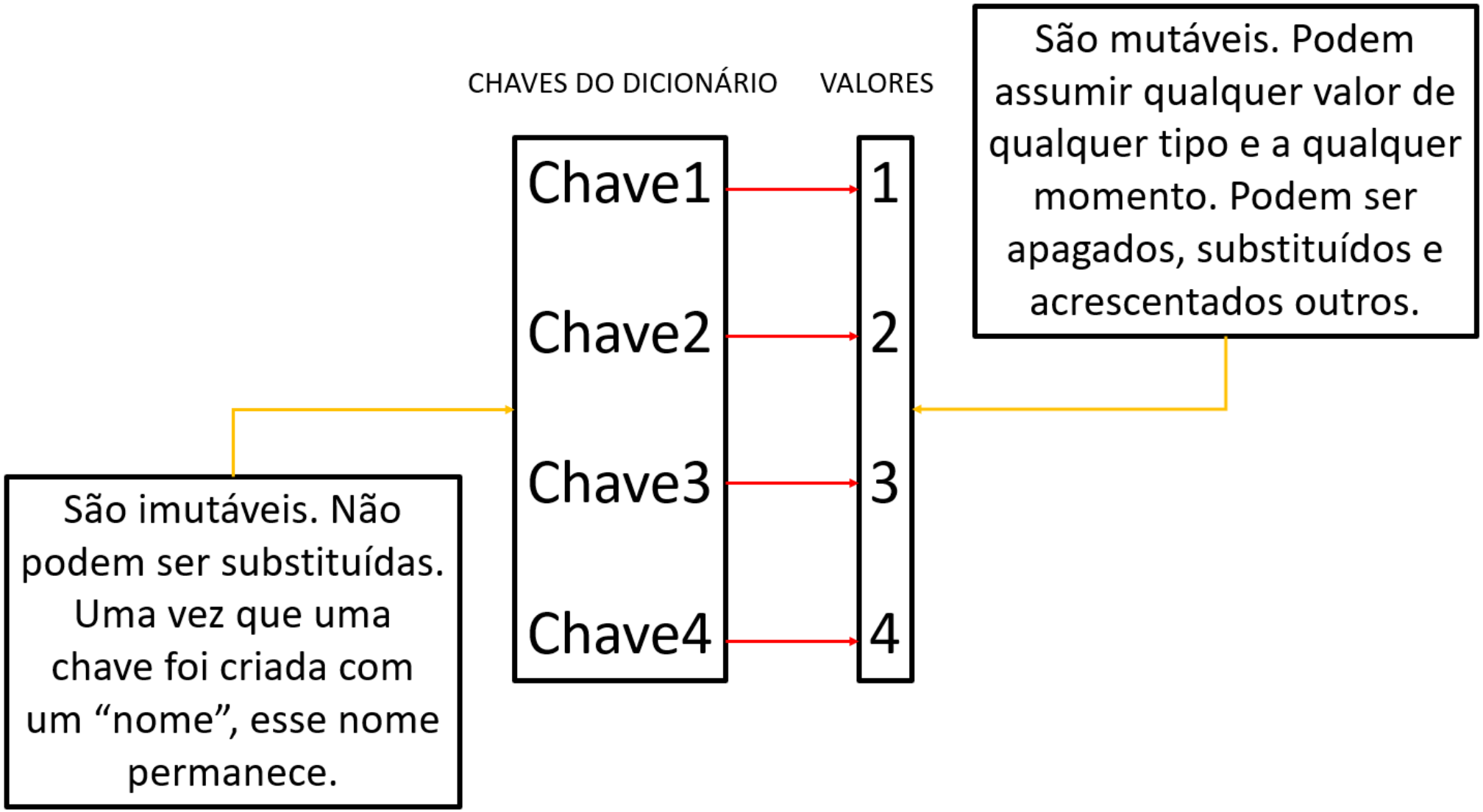
Dicionários:

```
>>> # Outra forma:

. . .
>>> dicionario = dict()
>>> dicionario['chave1'] = 1
>>> dicionario['chave2'] = 2
>>> dicionario['chave3'] = 3
>>> dicionario
{'chave1' : 1, 'chave2' : 2, 'chave3' : 3}
```

UNIDADE 6

Dicionários:



UNIDADE 6

Dicionários:

Alguns métodos e funções geralmente utilizadas com dicionários:

```
>>> tel = {'John' : 456789, 'Peter' : 784785, 'Vini' : 987584, 'James' :
142365, 'Mary' : 456987}
>>> # Adicionar um novo número à agenda:
. . .
>>> tel['Paul'] = 101010
>>> tel
{'John': 456789, 'Peter': 784785, 'Vini' : 987584, 'James': 142365, 'Mary':
456987, 'Paul': 101010}
```

UNIDADE 6

Dicionários:

Alguns métodos e funções geralmente utilizadas com dicionários:

```
>>> # Verificar se existe uma chave no dicionário:
. . .
>>> 'Paul' in tel
True
>>>
>>> # Retirar um telefone da agenda:
. . .
>>> del tel['James']
>>> tel
{'John': 456789 , 'Peter': 784785 , 'Vini' : 987584 , 'Mary': 456987 , 'Paul': 101010}
```

UNIDADE 6

Dicionários:

Alguns métodos e funções geralmente utilizadas com dicionários:

```
>>> # Método que mostra os itens com suas chaves de um dicionário:
. . .
>>> tel.items()
dict_items([('John', 456789), ('Peter', 784785), ('Vini', 987584), ('Mary', 456987),
('Paul’, 101010)])
>>>
>>> # Método que mostra todas as chaves de um dicionário:
. . .
>>> tel.keys()
dict_keys(['John', 'Peter', 'Vini', 'Mary', 'Paul'])
```

UNIDADE 6

Dicionários:

Alguns métodos e funções geralmente utilizadas com dicionários:

```
>>> # Método que mostra todos os valores presentes em um dicionário:
. . .
>>> tel.values()
dict_values([456789 , 784785 , 456987 , 101010])
>>>
>>> # A função builtin "len()" retorna o tamanho da lista (neste caso ,
o número de chaves):
. . .
>>> print(len(tel))
4
```

Exercício de Revisão: Unidade 6.4

Crie um programa que armazena um grupo de 3 pessoas contendo o número de telefone de cada uma e a idade de cada uma e depois mostre todas as informações e a informação apenas de uma pessoa.

- Obs.: Utilize um dicionário.

EXERCÍCIO EXEMPLO.

Exercício de Revisão: Unidade 6.4

```
agenda = {'Fulano' : [998754638, 19],
          'Ciclano' : [789456123, 20],
          'Beutrano' : [123456789, 19]}
print(agenda)
print(agenda['Fulano'])
```

UNIDADE 6

Matrizes:

Em Python, não existe uma função builtin que trabalhe com matrizes. Entretanto, é relativamente fácil e intuitivo a criação dessas. Nesse caso, a melhor opção para se trabalhar com matrizes de uma forma segura e sem ter que reinventar a roda é através das inúmeras bibliotecas e módulos disponíveis para esse fim.

Dessa forma, destaca-se a biblioteca **“NumPy”**, que se constitui como uma biblioteca projetada com códigos de alta performance.

UNIDADE 6

Matrizes:

Contudo, se o objetivo é programar uma matriz da forma como se costuma, então o Python simplifica essa situação pelo fato de que toda matriz é uma lista cujos elementos são outras listas. Isto é, para a criação de uma matriz “n” por “n” (n X n), basta que seja criada uma lista com “n” elementos e dentro de cada um seja criada outra lista com “n” elementos.

Existem diversas formas de se fazer isso. Abaixo é ilustrada uma dessas formas juntamente com o resultado impresso na tela.

```
mat = []
linhas = 5
colunas = 5
for i in range(linhas):
        lin = []
        for j in range(colunas):
                lin.append(int(i))
        mat.append(lin)
for i in mat: print(i)
```

```
## RESULTADO:
[0, 0, 0, 0, 0]
[1, 1, 1, 1, 1]
[2, 2, 2, 2, 2]
[3, 3, 3, 3, 3]
[4, 4, 4, 4, 4]
```

UNIDADE 6

Matrizes:

Contudo, se o objetivo é programar uma matriz da forma como se costuma, então o Python simplifica essa situação pelo fato de que toda matriz é uma lista cujos elementos são outras listas. Isto é, para a criação de uma matriz “n” por “n” (n X n), basta que seja criada uma lista com “n” elementos e dentro de cada um seja criada outra lista com “n” elementos.

Existem diversas formas de se fazer isso. Abaixo é ilustrada uma dessas formas juntamente com o resultado impresso na tela.

```
# Utilizando compreensão de listas (List Comprehension)
mat1 = [[i for j in range(5)] for i in range(5)]
for i in mat1: print(i)
```

RESULTADO:

[0, 0, 0, 0, 0]
[1, 1, 1, 1, 1]
[2, 2, 2, 2, 2]
[3, 3, 3, 3, 3]
[4, 4, 4, 4, 4]

UNIDADE 6

Matrizes:

Outro fato sobre matrizes em Python é que, por ser uma lista de outras listas, todos os métodos e funções das listas podem ser utilizados em matrizes (de forma correta). Cada elemento pode ser obtido a partir da posição bidimensional matriz[linha][coluna].

Exercício de Revisão: Unidade 6.5

Desenvolva um algoritmo que leia os elementos de uma matriz A de ordem 4 (4x4) e também de uma matriz B de mesma ordem, gere e imprima uma matriz com a soma dos elementos de A com B.

Exercício de Revisão: Unidade 6.5

```
A = []
B = []
C = []
for i in range(4):
    lin = []
    for j in range(4):
        lin.append(int(input('Elemento: ')))
    A.append(lin)
for i in range(4):
    lin = []
    for j in range(4):
        lin.append(int(input('Elemento: ')))
    B.append(lin)
for i in range(4):
    lin = []
    for j in range(4):
        lin.append(A[i][j] + B[i][j])
    C.append(lin)

for i in A: print(i)
for i in B: print(i)
print('\n')
for i in C: print(i)
```

UNIDADE 7 – Funções:

1.Funções

UNIDADE 7

Funções:

De acordo com Downey (2015), uma função é uma sequência nomeada de instruções que executa uma determinada operação, sendo criada a partir de um nome e de instruções que a compõe, possibilitando ser utilizada posteriormente.

UNIDADE 7

Funções:

De acordo com Borges (2010), as funções, em Python:

- **Podem retornar ou não objetos;**
- **Aceitam Doc Strings;**
- **Se não for passado o parâmetro será igual ao default definido na função. Portanto aceitam parâmetros default;**
- **Aceitam que os parâmetros sejam passados com nome. Neste caso, a ordem em que os parâmetros foram passados não importa;**
- **Tem namespace próprio (escopo local), e por isso podem ofuscar definições de escopo global;**
- **Podem ter suas propriedades alteradas (geralmente por decoradores).**

UNIDADE 7

Funções:

```
def nomeDaFuncao(parametro):
        """Esta é a forma de se declarar
        funções em Python. Este texto é um
        Doc String."""
        print('Esta função recebeu ', parametro , ' como parâmetro. ‘
                '\nO tipo deste parâmetro , é: ‘,
                type(parametro))
        return '\nApostila de Python é top!'
print(nomeDaFuncao('string'))
```

UNIDADE 7

Funções:

```
## RESULTADO:
Esta função recebeu string como parâmetro.
O tipo deste parâmetro , é: <class 'str'>
Apostila de Python é top!
```

UNIDADE 7

Funções:

A partir do momento em que uma função é criada, é possível utilizá-la dentro de outras funções e, inclusive, dentro dela mesma.

Esse conceito é base para recursividade de funções. A seguir é mostrado um exemplo de uma função recursiva que calcula o fatorial de um número “n”.

UNIDADE 7

Funções:

```
def fatorial(numero):
    if type(numero) is not int:
        return '\nO número deve ser um inteiro!’
    if numero <= 0:
        return 1
    else:
        return numero * fatorial(numero-1)
```

UNIDADE 7

Funções:

É interessante pontuar que, em muitos casos, é do interesse do programador criar funções que não retornem nada, apenas façam algo ou imprimam algo na tela. Nesse caso, essas funções são chamadas de funções nulas, segundo Downey (2015). Esse tipo de função é muito utilizado em **OOP** (Programação Orientada a Objetos), nos "getters and setters".

UNIDADE 7

Funções:

Para criar uma função que não retorne nada, basta escrever no fim da função **“return None”**, ou simplesmente não escrever **“return”**.

```
def naoRetornaNada_ApenasPrinta():
    print('\nApenas printa qualquer coisa')
naoRetornaNada_ApenasPrinta()
```

UNIDADE 7

Funções:

Além disso, os argumentos das funções também podem ser listas, tuplas e/ou dicionários (além dos valores default). Para listas, basta escrever “*” na frente do nome do argumento na função; neste caso, todos os parâmetros presentes após o asterisco serão considerados como uma lista de elementos.

Porém, o tipo de return será uma tupla.

UNIDADE 7

Funções:

```
def recebe_lista(*lista):
        return lista
a = recebe_lista(0, 5, 8, 7, 6, 4)
print(a)
print('TIPO: ', type(a))

## RESULTADO:
(0, 5, 8, 7, 6, 4)
TIPO: <class 'tuple'>
```

UNIDADE 7

Funções:

Para dicionário, basta escrever **“**”** na frente do nome do argumento na função. Neste caso, cada valor no argumento deve ser indicado utilizando o operador **“=”** para indicar chaves e valores. Abaixo é ilustrada essa afirmação.

```
def recebe_dicionario(** dic):
        return dic
a = recebe_dicionario(a=1, b=2, c=3)
print(a)
print('TIPO: ', type(a))
## RESULTADO:
{'a': 1, 'b': 2, 'c': 3}
TIPO: <class 'dict'>
```

UNIDADE 7

Funções:

Por fim, faz-se necessário destacar que o fluxo da programação em Python sempre é interrompido no momento que uma função é chamada. Quando isso acontece, ele interrompe a sequência do código, executa a função invocada e, após o término da função, retorna a sequência do ponto em que havia parado.

Por esse motivo, é necessário que as funções sejam previamente criadas. Assim, recomenda-se que essas estejam localizadas após a importação das bibliotecas e/ou módulos utilizados, no início do código.

Exercício de Revisão: Unidade 7

Faça um programa que contenha uma função que receba como parâmetro uma string e um caractere e mostre na tela quantas vezes o caractere ocorre na string.

Exercício de Revisão: Unidade 7

```
def func(string, caractere):
    return string.count(caractere)
print(func('abcabc', 'a'))
```

Python 101 para Engenheiros

Obrigado!