Python para Ciência de Dados

Aula 02

SENAI CETIQT SENAI

Currículo: Profa. Edileusa de Estefani do Prado

FORMAÇÃO:

- Mestranda em Métodos Quantitativos - FEA/USP – 2019-2022.
- MBA em Analytics – Big Data - FIA – 2015/2016 – Turma 1.
- Disciplina de Banco de dados avançados - UNICAMP – 2005.
- Pós-Graduação em Consultoria em Internet - FASP – 2000/2001.
- Graduação em Ciências da Computação - UNESP – 1989/1993.

QUALIFICAÇÕES:

- Experiência profissional de 26 anos no mercado, tendo atuado como Data Scientist, DW DBA, BI, ETL, AD e Analista de Sistemas em empresas: Banco Itaú, Equifax do Brasil, Banco ABN Amro, IBM, Serasa Experian, Unisys Brasil, e outras.
- Professora dos cursos de MBA e Pós-Graduação em Analytics e Big Data pela FIA.
- Professora do curso de Pós-Graduação em Big Data pelo Senac – SJRP – SP.
- Data Scientist Consultant

CONTATO:

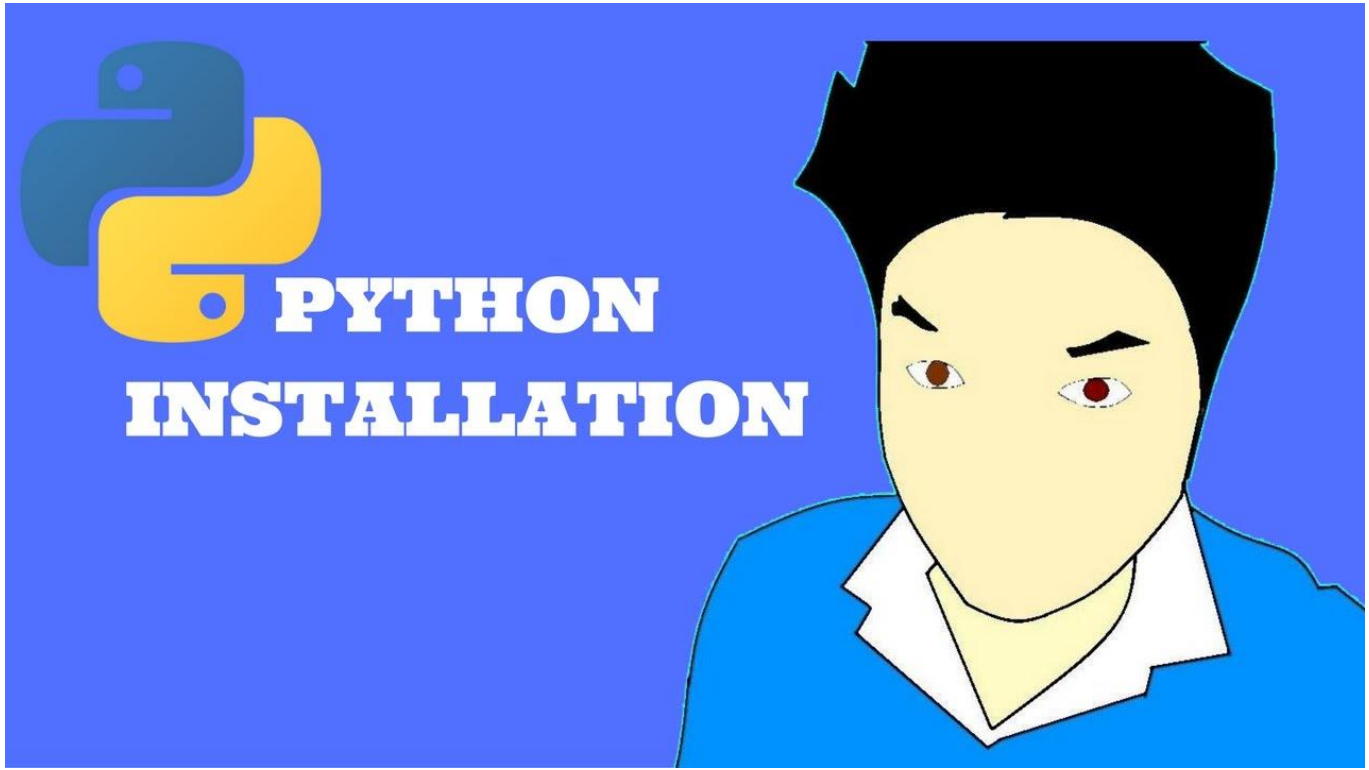
- E-mail : edileusa.estefani@gmail.com
- LinkedIn : https://www.linkedin.com/in/edileusa-est%C3%A9fani-prado-027a957/

Agenda

1. Objetivo
2. Rever instalação do Anaconda3
3. Primeiros passos na utilização do Anaconda3
4. Tipos de dados básicos, operadores e variáveis
5. Comparação de Dados e Condicional
6. Laços – (for – while)
7. Laços – (for – loop)
8. Strings e Slices de Strings

Objetivo

- Auxiliar os alunos a efetuarem a instalação do software Anaconda3 versão de novembro/2021.
- Primeiros passos na utilização do Anaconda3 e Notebook Jupyter
- Apresentar aos alunos, principalmente a quem tem pouca ou nenhuma experiência com a linguagem Python, os principais conceitos sobre:
  - Tipos de dados básicos, operadores e variáveis em Python
  - Comparação de dados e condicional (incluindo condicional aninhada)
  - Laços – (for – while)
  - Laços – (for – loop)
  - Strings e Slices de Strings


PYTHON
INSTALLATION

Tipos de dados básicos, operadores e variáveis

- Os principais operadores aritméticos usados em Python são:

+ : Soma

- : Subtração

***** : Multiplicação

/ : Divisão

****** : Exponenciação

% : Módulo (resto da divisão)

Tipos de dados básicos, operadores e variáveis

```
>>> 2 + 2 # operação de soma de inteiros

>>> 2 + 2.0 # soma de inteiro com float resultado float

>>> 5 - 3 # subtração de inteiros

>>> 5.0 - 3 # subtração de float com inteiro resultado float

    5 - 3.0

>>> 4 * 5 # multiplicação de dois inteiros resultado inteiro

>>> 4 * 5.0 # multiplicação de inteiro com float resultado float
```

❑ Utilizando o Python como uma calculadora

```
>>> 20 / 5 # o resultado da divisão de 2 inteiros usando apenas 1 / é sempre

>>> 20 // 5 # No Python 3 para forçar a divisão de 2 inteiros permanecer resultado inteiro use

>>> 20.0 // 5.0 # com floats o resultado da divisão é

>>> 20.0 // 5

>>> 20.0 / 5.0

>>> 17 / 3 # clássica divisão de inteiros com / resultado será

>>> 17 // 3 # com // o resultado retornará
```

❑ Utilizando o Python como uma calculadora

```
>>> 17 % 3 # resto da divisão entre dois inteiros o resultado é

>>> 17.0  % 3 # resto da divisão entre um float e um inteiro o resultado é

>>> 5 ** 2 # exponenciação entre dois inteiros o resultado é um

>>> 5.0 ** 2 # exponenciação entre um float e um inteiro o resultado é


>>> 50 - 5 * 6 # a precedência das operações aritméticas funciona da mesma forma sem o uso dos parenteses

>>> 50 - 5 * 6 / 3

>>> (50 - 5 * 6) / 4

>>> 50 - 5 * 6 / 4
```

❑ Sentenças em Python

```
Olá, meu nome é Maria
>>> print('Olá, meu nome é Maria') # correto usando aspas simples
Olá, meu nome é Maria
>>> print("Olá, meu nome é Maria") # correto usando aspas duplas
Olá, meu nome é Maria
>>> print("Olá, meu nome é 'Maria'.") # aspas simples dentro de aspas duplas - começou com aspas duplas fecha com aspas duplas

>>> print('Olá, meu nome é "Maria".') # aspas duplas dentro de aspas simples - começou com aspas simples fecha com aspas simples

>>> print('Olá, meu nome é \'Maria.\'') # aspas simples dentro de aspas simples

>>> print("Olá, meu nome é \"Maria.\"") # aspas duplas dentro de aspas duplas

>>>
```

\ → permite avançar para o próximo caracter.

❑Variáveis

- ❑ Uma variável é uma localização que armazena algum dado na memória do computador, como por exemplo, números ou strings e a localização da variável é associada com um nome (identificador).
- ❑ Para acessar a variável no programa eu preciso dar um nome a ela.

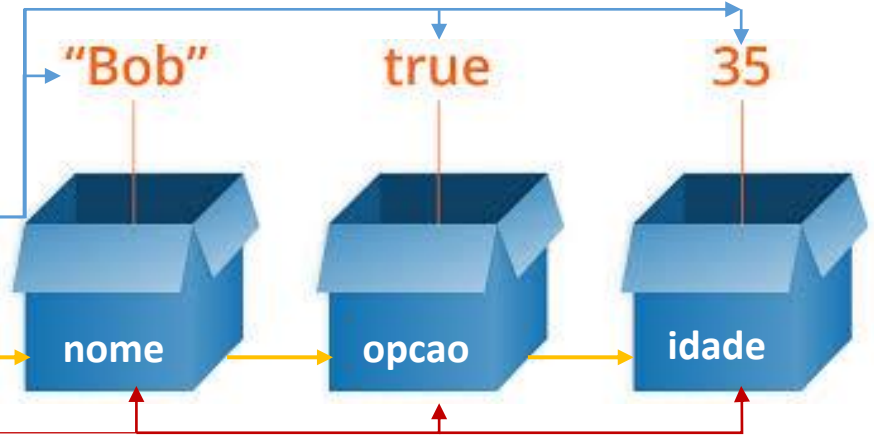
❑Atribuição

Um comando de **atribuição** tem a seguinte forma:

Variável = expressão

- Significado:

1. O valor da expressão é calculado;
2. A variável é criada e nomeada;
3. A variável passa a fazer referência, ou seja guardar o valor.



❑Variáveis

Exemplos:

score = 7.5 → a variável denominada **score** armazena o valor numérico 7.5

nome = “Maria Aparecida” → o conteúdo da variável **nome** é o valor da string “Maria Aparecida”

ativo = True

idade = 33

Idade = 33

Observação: Notar que **idade** é diferente de **Idade** no Python. Python é **case sensitive.**

```
num = 15
soma = 0
soma = soma + num  # soma será igual a 15
```

❑Variáveis

Segue abaixo as regras para nomeação de variáveis do Python:

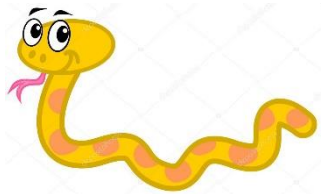
❑ Pode:

✓ Devem iniciar com uma letra (ou um underscore _)

✓ Podem ter até 256 caracteres no total

✓ Podem incluir letras, números, underscores, $, etc.

❑Não pode:



✓ Começar com número → 1nome

✓ Conter espaços → minha idade

✓ Conter símbolos matemáticos: + - * / parênteses → ativo+passivo (score)

❑Variáveis

Temos as palavras reservadas à linguagem que também não podem ser utilizadas como variáveis.

```
and       as        assert    break
class
continue  def       del       elif
else
except    exec      if        import
in
is        lambda    not       or        pass
print     raise     return    try
while
yield     True      False     None
```

❑A função input()

A função input() **recebe os dados** ou **valores** de um usuário através do teclado.

Formato típico do comando: **variavel = input("Prompt")**

O programa pára e espera pela digitação de algum texto seguido do ENTER. "Prompt" é opcional.

Ex.:
```
nome = input("Qual é o seu nome? ")
print(nome, ", me fale sobre você.")
```

O valor que o usuário fornece e que será retornado pelo input() **é sempre uma string (não um número).**

Se houver o operador + entre strings, elas serão concatenadas (“grudadas”).

❑A função int()

Em Python, a função int() converte um dado string para um número inteiro.

Exemplo:

```
>>> a = int("456")
>>> print('Valor inteiro eh:', a)
Valor inteiro eh: 456
>>> a = a + 4
>>> print('Soma de dois inteiros:', a)
Soma de dois inteiros: 460
```

Estrutura de um programa

Para fins didáticos, utilizaremos a seguinte estrutura para nossos programas em Python.

```
def main():
    # comandos
    ...


#------
# a linha a seguir inicia a execução do programa
main()
```

A função print() é utilizada para o programa exibir mensagens.

Exemplo: Tentativa 1: execute o programa abaixo e veja o que acontece.

```
1 def main():
2     a = 3
3     b = 4
4     soma = a + b
5     print("A soma de", a, "+", b, "eh igual a", soma)
6
7 main()
8
9
```

Tentativa 2: execute o programa abaixo e veja o que acontece.

```
1 def main():
2     a = input("Digite o primeiro numero: ")
3     b = input("Digite o segundo numero: ")
4     soma = a + b
5     print("A soma de", a, "+", b, "eh igual a", soma)
6
7 main()
8
9
```

❑A função print()

Tentativa 3: execute o programa abaixo e veja o que acontece.

```
1  def main():
2      a = int(input("Digite o primeiro numero: "))
3      b = int(input("Digite o segundo numero: "))
4      soma = a + b
5      print("A soma de", a, "+", b, "eh igual a", soma)
6
7  main()
8
9
```

- Exercícios 1 - Calcule quanto dinheiro devem te pagar por trabalhar em um emprego. Durante as primeiras 40 horas, você deve ser pago a uma taxa horária. Qualquer hora acima de 40 deve ser paga a uma hora e meia da taxa, exceto se você trabalhou no final de semana, pois daí devem te pagar o valor da hora em dobro. Quanto deverá receber e qual o valor do seu fgts hipotético, segundo a fórmula abaixo? Segue abaixo ajuda para os cálculos:

```
rate = 100.00            # valor da hora trabalhada
totalHours = 70          # total de horas semanais trabalhadas
regularHours = 40        # total de horas semanais regulares
weekendHours = 10  # horas trabalhadas no final de semana
overTimeHours = totalHours - regularHours - weekenHours  # horas extras normais
pay1 = (rate * regularHours) + ((rate * 1.5) * overTimeHours)
pay2 = (rate * 2.0) * weekendHours
fgts2 = pay12 + pay22 # Fórmula do fgts hipotético
Imprima('Por trabalhar', totalHours, ' horas, eu deveria receber: ', pay1 + pay2, ' com fgts = ' fgts)
```

Comparação de Dados e Condicional

Expressões Lógicas

Condições ou expressões lógicas *(boolean expressions)* são expressões cujo valor é **verdadeiro** (=``**True**`` em Python) ou **falso** (=``**False**`` em Python) e usam os **operadores relacionais**:

> (maior);
>= (maior ou igual);
< (menor);
<= (menor ou igual);
== (igual); ou
!= (diferente).

Valores booleanos

Em Python, **uma variável pode assumir** valor booleano **True** (verdadeiro) ou **False** (falso). Esses valores são **úteis** para **representar**, por exemplo, **o resultado de uma comparação**. Experimente faça a tentativa abaixo.

```
1  a = 3
2  b = 4
3  c = a < b   # c recebe o valor da comparação a < b
4  d = a > b   # d recebe o valor da comparação a > b
5  e = a == b  # e recebe o valor da comparação a == b
6
7  print("Valor de c: ", c)
8  print("Valor de d: ", d)
9  print("Valor de e: ", e)
10
11
```

```
>>> c
True
>>> d
False
>>> e
False
```

Valores booleanos

Cuidado em Python, = e == não são a mesma coisa.

Um é usado **para atribuição de valor a uma variável (=)** e o outro para **comparação de valores iguais (==)**.

```
>>> a = 2
>>> b = 4
>>> a = b
>>> a == b
```

Aqui a recebeu o valor que está em b

Comparando a com b

Operadores e expressões lógicas

Operador and : Dados dois valores booleanos A e B, **o operador lógico and** resulta em **True apenas quando A e B foram ambos True**, e retorna **False caso contrário**.

A tabela abaixo mostra o resultado de `and` para todas as combinações de A e B.

and	A = True	A = False
B = True		
B = False		

Operadores e expressões lógicas

Operador or : Dados dois valores booleanos A e B, **o operador lógico or** resulta em **False apenas quando A e B foram ambos False**, e retorna **True caso contrário**.

A tabela abaixo mostra o resultado de `or` para todas as combinações de A e B.

or	A = True	A = False
B = True		
B = False		

Operadores e expressões lógicas

Operador not : O operador lógico **not** ***muda o valor de seu argumento***, ou seja:

- **not True** é **False**, e
- **not False** é **True**.

Funções para conversão de valores: int( ), float( ), str( )

A **função int** converte para **inteiro,** tanto um **decimal** quanto um **string (diferente de letras)**. Decimais são **truncados**.

A **função float** transforma um **inteiro**, um **decimal** ou um **string em float**.

A **função str** transforma seus argumentos em uma **string**.

❑ Exercícios Homework: Façam as tentativas para treinar

```
print(3.14, int(3.14))
print(3.9999, int(3.9999))        # Isto não arredonda para o inteiro mais próximo
print(3.0,int(3.0))
print(-3.999,int(-3.999))         # Observe que o resultado está mais próximo de zero

print("2345",int("2345"))         # examina um string para produzir um int
print(17,int(17))                 # int também funciona sobre inteiros
print(int("23garafas"))


```

```
print(float("123.45"))
print(type(float("123.45")))


```

```
print(str(17))
print(str(123.45))
print(type(str(123.45)))


```

Execução condicional If

```
if condiçao:
    |
    | bloco de comandos
    |
```

Executa o **bloco de comandos** se a condição **for verdadeira**

Faça experimentos com o trecho de código a seguir.

```
1 x = float(input("Digite um numero: "))
2 if x < 5:
3     print(x, "< 5")
4
5 print("Termino do teste.")
6
7
```

Entre com Valores: 0 e 7

Execução alternativa If-else

```
if condição:
    |
    | bloco de comandos 0
    |
else:
    |
    | bloco de comandos 1
    |
```

Se a condição **for verdadeira** executa o **bloco de comandos 0** se a condição **for falsa** executa o **bloco de comandos 1.**

Faça experimentos com o trecho de código a seguir.

```
1 x = float(input("Digite um numero: "))
2 if x < 5:
3     print(x, "< 5")
4 else:
5     print("5 <=", x)
6
7 print("Termino do teste.")
8
9
```

Entre com Valores: 0 e 7

Execução condicional em cadeia If-elif-else

As condições são testadas em ordem, uma após outra.

Apenas o bloco de comandos correspondente a 1ª condição que for verdadeira, será executado.

O bloco de comandos associados ao else serão executados apenas se todas as condições forem falsas.

Não é necessário que haja um else no final.

```
if condição_0:
    |
    | bloco de comandos 0
    |
elif condiçao_1:
    |
    | bloco de comandos 1
    |
elif condicão_2:
    |
    | bloco de comandos 2
    |

elif ...

elif condição_k:
    |
    | bloco de comandos k
    |
else:
    |
    | bloco de comandos k+1
    |
```

Execução condicional em cadeia If-elif-else

Faça experimentos com o trecho de código a seguir.

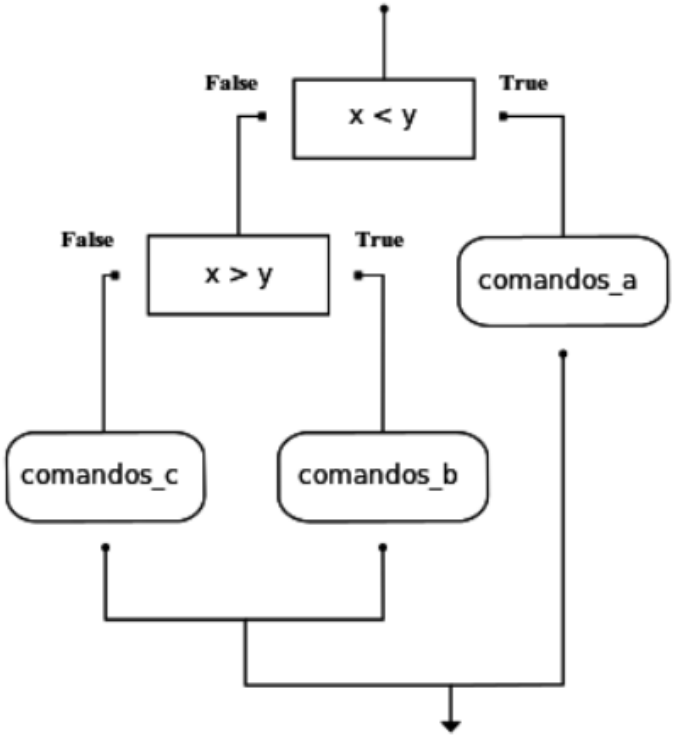
```
1  x = float(input("Digite um numero: "))
2  if x < 0:
3      print(x, "< 0")
4  elif x == 0:
5      print(x, "== 0")
6  elif x < 1:
7      print("0 < ", x,"< 1")
8  elif x < 2:
9      print("1 <=", x, "< 2")
10 elif x < 3:
11     print("2 <=", x, "< 3")
12 elif x < 5:
13     print("3 <=", x, "< 5")
14 else:
15     print("5 <=", x)
16
17 print("Termino do teste.")
18
19
```

Homework: Testem com os números -0.5, 0, 0.5, 1.5, 2.5, 3.5, 4.5, 7

Execução condicional aninhada

```
if x < y:
    print("x e' menor do que y.")
else:
    if x > y:
        print("x e' maior do que y.")
    else:
        print("x e y devem ser iguais.")
```

O fluxo de controle pode ser visto nesta ilustração do fluxograma.



Laços – (for – while)

While Loop

While

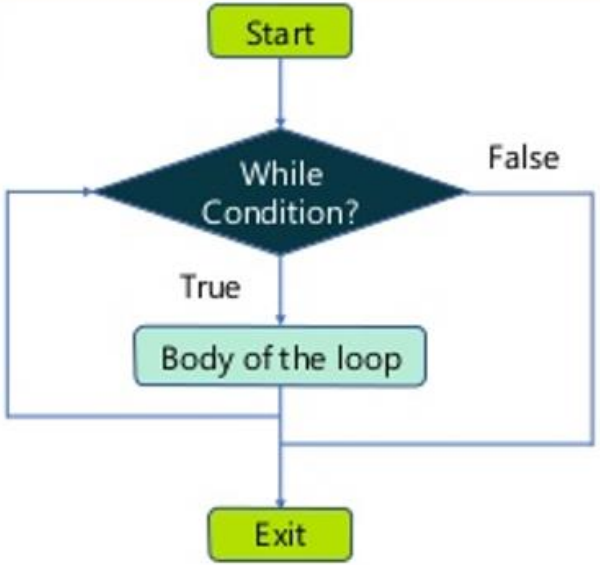
For

Nested

While loops are known as indefinite or conditional loops. They will keep iterating until certain conditions are met. There is no guarantee ahead of time regarding how many times the loop will iterate.

Syntax:

```
1 while expression:
2     statements
```



Sintaxe do comando while

```
while <condição>:
    # sequência de comandos executados no corpo do while
    <comando_1>
    <comando_2>
    ...
    <comando_n>
```

Exemplo

```
1                       # inicialização
2           fim  = 5        # número de iterações
3           cont = 0         # variável de controle
4
5           while cont < fim:
6               # faça alguma coisa, nesse caso, apenas imprima cont
7               print("Iteracao numero: ", cont)
8
9               cont = cont + 1     # variável de controle precisa ser
10                                  # atualizada para garantir o fim do while
```

Exercicio 1

```
>>> temperature=0
>>> while temperature<35:
        temperature=temperature+5
        if temperature<15:
                print(str(temperature)+" degrees is cold")
        elif temperature<25:
                print(str(temperature)+" degrees is warm")
        else:
                print(str(temperature)+" degrees is hot")
```

Exercicio 2

```
>>> from random import randint
>>> number = randint(1,100)
>>> prediction = 0
>>> while prediction != number:
        prediction = int(input("New number: "))
        if prediction > number :
                print("Number too large")
        elif prediction < number :
                print("Number too small")
        else:
                print("Congratulation. You made it!")
```

Laços – (For Loop)

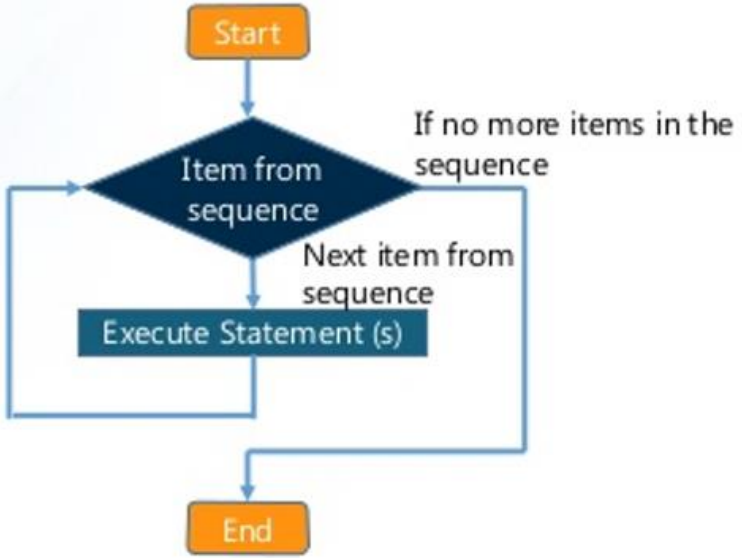
For Loop

For Loop

Repeats a statement or group of statements while a given condition is TRUE. It tests the condition before executing the loop body.

Syntax:

```
1 for iterating_var in sequence:
2     statements
```



❑ FOR

Como é muito comum percorrer listas, do início ao fim, para processar cada elemento, podemos utilizar o comando `for` das seguintes maneiras:

```
primos = [2, 3, 5, 7, 11, 13]
for elem in primos:
    print( elem )
```

Homework

```
for amigo in ["Joe", "Amy", "Brad", "Angelina", "Zuki", "Thandi", "Paris"]:
    print("Ola ", amigo, "  Por favor venha a minha festa no sabado!")
```

```
for i in range(4):
    # Executa o corpo com i = 0, depois 1, depois 2, depois 3
for x in range(10):
    # x recebe um valor de [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] de cada vez
```

❑ Exercício

```
>>> animals = ['human', 'monkey', 'cat', 'dog']

>>> for i in range(len(animals)):
        print(i, animals[i])
```

❑ Função em Python

```
def "nome da função"("parâmetros"):
    '''
    docstring contendo comentários sobre a função.
    Embora opcionais são fortemente recomendados. Os comentários
    devem descrever o papel dos parâmetros e o que a função faz.
    '''
    # corpo da função
    |
    | bloco de comandos
    |
```

❑ Esqueleto de um programa Python

```
# função  principal
def main():
    '''
    Função principal, será a primeira a ser executado e
    será a responsável pela chamada de outras funções que
    por sua vez podem ou não chamar outras funçoes que
    por sua vez ...
    '''
    # corpo da função main
    |
    | bloco de comandos
    |

# Declaração das funções
def f...
    '''
    docstring da função f
    '''
    # corpo da função f
    |
    | bloco de comandos
    |

def g...
    '''
    docstring da função g
    '''
    # corpo da função g
    |
    | bloco de comandos
    |

[...]

# início da execução do programa
main() # chamada da função main
```

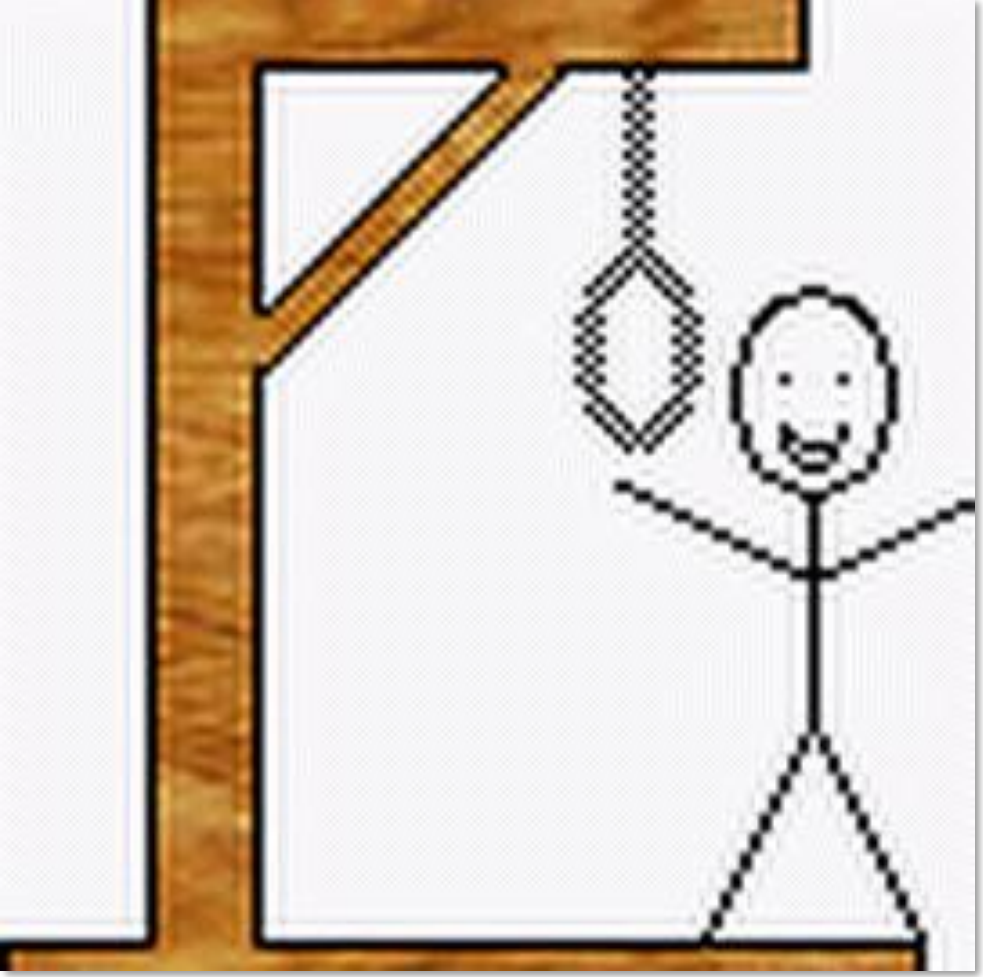
```
def calculateAverage(param1, param2, param3, param4):
        total = param1 + param2 + param3 + param4
        average = total / 4.0
        print (‘Media do valor e: ‘, average)

calculateAverage(2, 3, 4, 5)
```

```
def calculateAverage(param1, param2, param3, param4):
        total = param1 + param2 + param3 + param4
        average = total / 4.0


media = 0
media = calculateAverage(2, 3, 4, 5)
print ‘Media do valor e:’, media
```

O que está faltando nesta função para o retorno do valor?

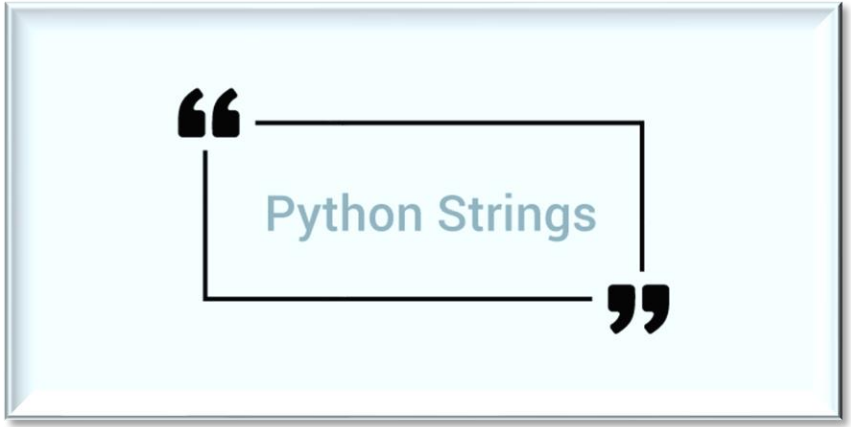
❑ Execute a função abaixo - Homework

```
def fatorial(k):
    '''(int) -> int

    Recebe um inteiro k e retorna o valor de k!

    Pre-condicao: supoe que k eh um numero inteiro nao negativo.
    '''

    k_fat = 1
    cont = 1
    while cont < k:
        cont += 1        # o mesmo que cont = cont + 1
        k_fat *= cont    # o mesmo que k_fat = k_fat * cont

    return k_fat

# testes
print("0! =", fatorial(0))
print("1! =", fatorial(1))
print("5! =", fatorial(5))
print("17! =", fatorial(17))
```



SENAI CETIQT SENAI TEIA

Strings e Slices de Strings



- Uma string é um **texto** entre aspas " ou entre apóstrofo '

Exemplo:

```
>>> variavel_string1 = "Nome: Maria Aparecida"
>>> variavel_string2 = "Endereço: Rua das Flores 345 - Vila Sônia - São Paulo"
>>> print(variavel_string1, variavel_string2)

>>> print(variavel_string1, '\n', variavel_string2)
```

Tipos de valores

- Um valor pode ser um número inteiro (classe int);
- Um número float (classe float) ou ainda
- Um texto (classe str)
- Usando a função type do Python, podemos obter o tipo ou classe de um dado valor.

Exemplos:

```
>>> type("Aniversário de São Paulo, 25 de janeiro")

>>> type('IOT')

>>> type(23)

>>> type('23')
```

Strings

Ene

Ex.:

```
>>> type(0.5)
>>> type("0.5")
>>> type("*")
>>> type(True)
>>> type(False)
>>> type(2,500)


>>> type(2500)
```

Observação: Na função type insira o valor numérico sem o uso da vírgula.

Operações com strings

- Você **não pode** executar operações matemáticas em strings, mesmo que se pareçam com **números**.
- Para strings, você pode utilizar o operador **+** que significa **concatenação de strings**, ou seja, **concatenar strings,** (“grudar strings uma na outra”)

Ex.:

```
>>> fruta = 'banana'
>>> assada = 'com canela'
>>> print(fruta + assada)
bananacom canela
>>> assada = ' com canela' # dar um espaço antes para melhorar a impressão da string
>>> print(fruta + assada)
banana com canela
```

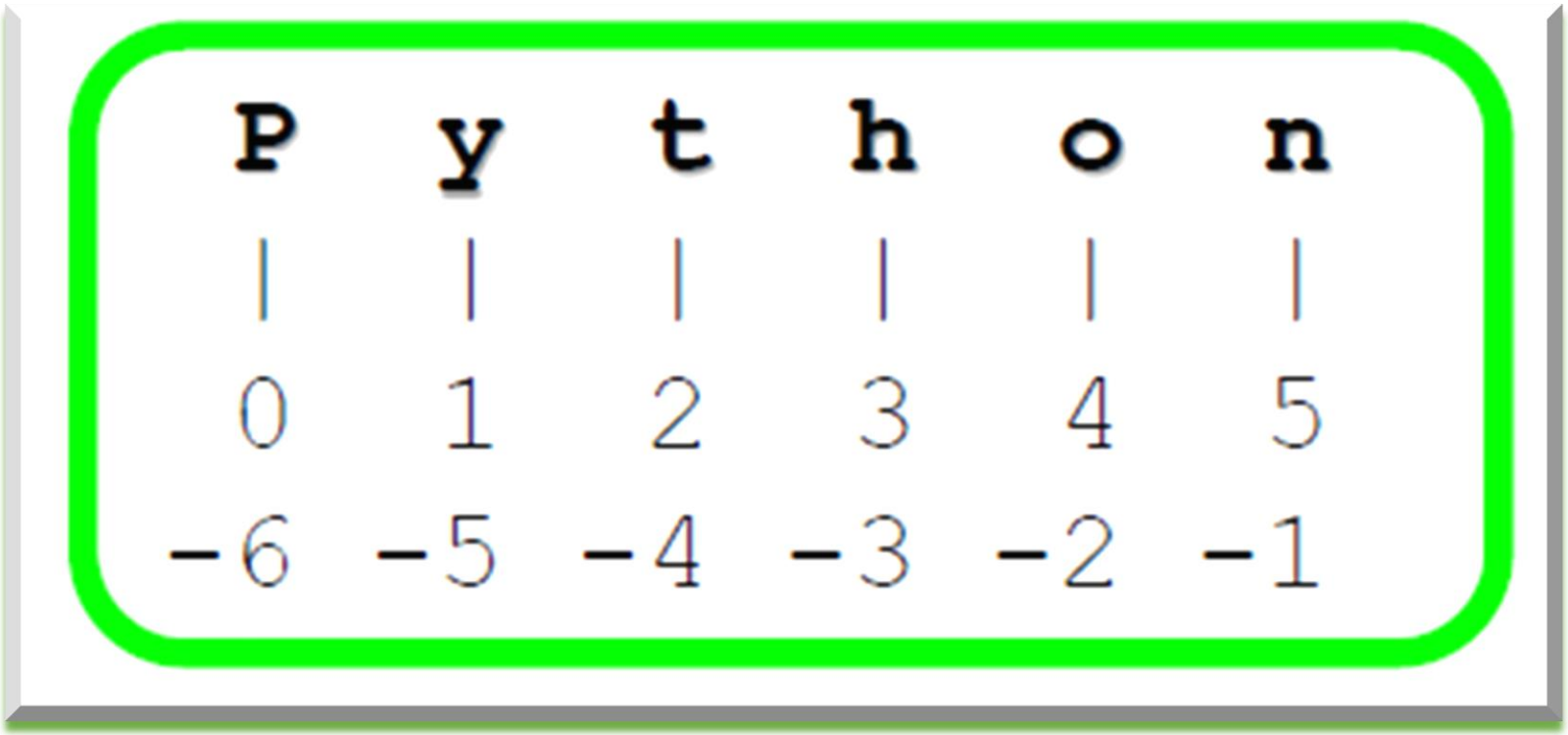
Operações com strings

- Para strings, você pode utilizar o operador * que significa repetição da string, ou seja, replicar a string *n* vezes, onde um operador é **string** e outro é um **inteiro**.

Exemplos:

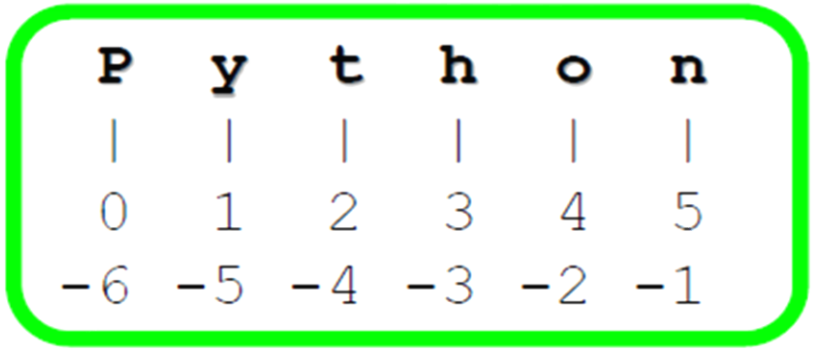
```
>>> print("Uau!"*3)
Uau!Uau!Uau!
```

Posicionais de uma string (Slices de Strings)



Posicionais de uma string

Exemplos:

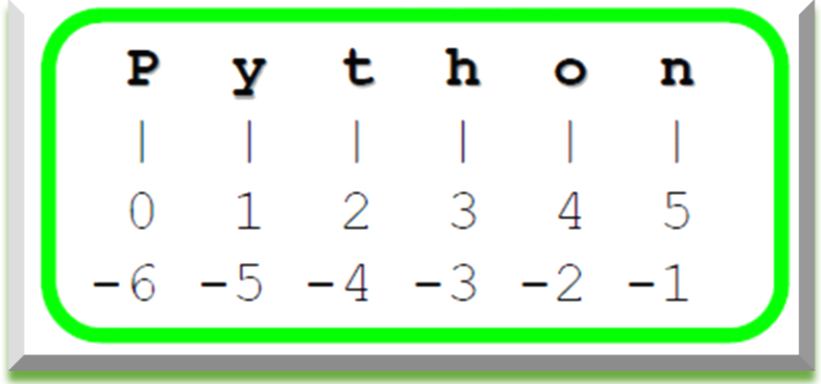



```
>>> word = 'Python'
>>>
>>> word[0]   # primeiro caracter da string na posição 0 (zero)
>>> word[1]  # segundo caracter da string na posição 1
>>>
>>> word[-1] # ultimo caracter da string
>>> word[-2] # penúltimo caracter da string
```

A posição em uma string inicia-se por zero.

Posicionais de uma string

Exemplos:




```
>>> word[0 : 2] # caracter partindo da posição 0 (incluída) até a posição 2 (excluída)

>>> word[2 : 5] # partindo da posição 2 (incluída) até a posição 5 (excluída)
```

```
>>> word[4 :] # partindo da posição 4 (incluída) até o final

>>> word[: 2] # partindo da ínicio até a posição 2 (excluída)

>>> word[-2 : ]  # partindo de -2 (inclusive) até o final
```

Posicionais de uma string

P	y	t	h	o	n
0	1	2	3	4	5
-6	-5	-4	-3	-2	-1

Exemplos:

word[25]

```
>>> word[ 25 ] # a palavra Python tem 6 caracteres apenas
Traceback (most recent call last):
  File "<pyshell#110>", line 1, in <module>
    word[ 25 ] # a palavra Python tem 6 caracteres apenas
IndexError: string index out of range
>>>
>>>
```

word[2] = ‘E’

Tentativa 01: Mostrar como criar um string a partir de um string vazio e efetuar a concatenação com outros strings. Abrir e executar o arquivo abaixo, usando o editor Spider do Anaconda.

```
def main():
    frase = ""    # string vazio
    frase = frase + "Esse string usa "
    meio  = '"apóstrofes" '
    fim   = "como demarcador. "
    frase = frase + meio + fim
    print("A frase: ", frase)
    print("Tem comprimento: ", len(frase))

main()
```

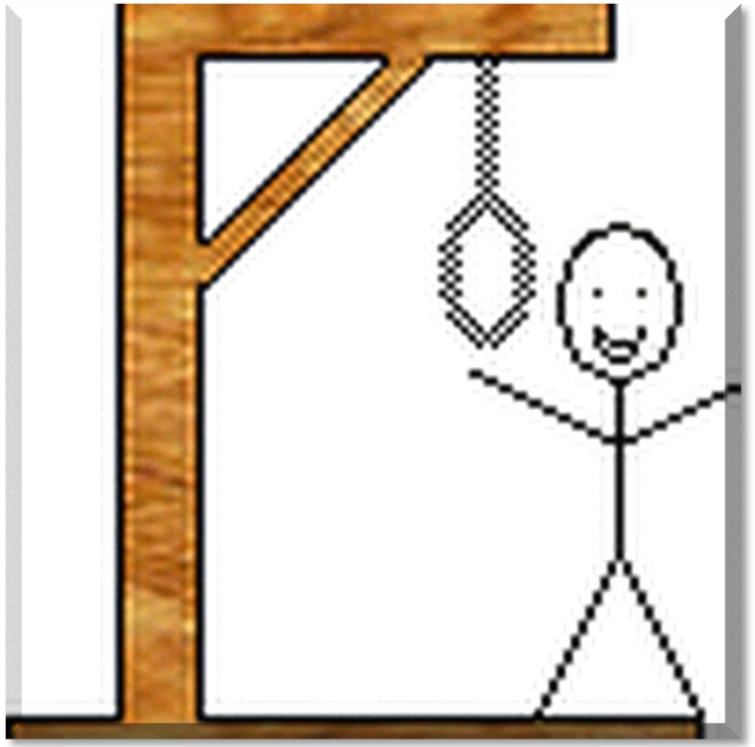
Arquivo: string_01.py

Tentativa 02: Duplicar o arquivo anterior, e após a exibição das mensagens, exibir uma terceira mensagem usando a variável frase e exibindo apenas a palavra demarcador. Duplicar e efetuar as modificações.

Arquivo: string_01.py

Salvar o novo arquivo com o nome String_02.py

Aplicação usando Strings: Jogo da Forca



Arquivo: jogo_forca.py

Abrir o arquivo: jogo_forca.py

Analisar o código e modificá-lo para imprimir a palavra secreta quando o jogador é enforcado.

Referências Bibliográficas

Sites:

- https://developers.google.com/edu/python/
- https://www.kdnuggets.com/2017/07/6-reasons-python-suddenly-super-popular.html
- http://letzgro.net/blog/creating-ai-using-python/
- https://dzone.com/articles/which-are-the-popular-languages-for-data-science

Livros:

- Curso Intensivo de Python, Novatec, Eric Matthes
- Introdução à Programação com Python, Novatec, Nilo Ney Coutinho Menezes, 2ª Revisão.
- Pense em Python, Novatec, Allen B. Downey
- Learn to Program with Python, Apress, Irb Kalb