Mineração de Dados

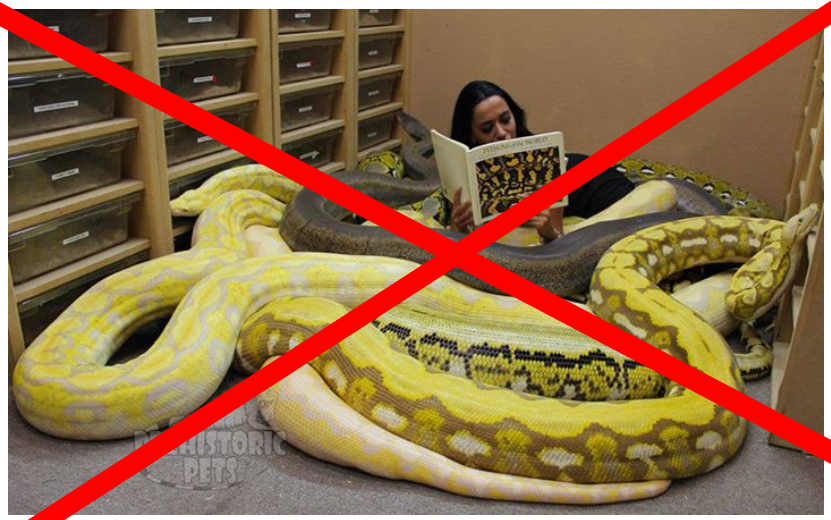
Introdução ao Python

Sumário

Introdução ao Python

Python

- Iniciada na década de 80 por Guido van Rossum, teve sua primeira versão lançada em 1991



- Interpretada
- Orientada a Objetos

Python

- Características [1]
  - Fácil de aprender
  - Fácil de ler
  - Possui uma quantidade grande de bibliotecas padrões
  - Possui um modo interativo
  - Portável
  - Estendível
  - Possui interface com diversos bancos de dados
  - Suporta aplicações gráficas
  - Escalável

[1] http://www.tutorialspoint.com/python

Ambiente

Ambiente

- Bibliotecas
  - numpy
  - scipy
  - matplotlib
  - pandas
  - scikit-learn

Python

- Esqueça declarações de tipos de variáveis
- Esqueça `begin` e `end`
- Esqueça { e }
- "Esqueça" ;
- Organização não é opcional
  - É obrigatório identar o código

Executando um Código em Python

- Execute o comando que segue

```
print("Olá, Python!")
```

- Crie um *script* `.py`

- Crie um *script* `.py` para ser diretamente executado
  - `chmod +x test.py & ./test.py`

```
#!/usr/bin/python3
print("Olá, Python!")
```

Linhas e Identação

- O código que segue está correto

```
if True:
    print("Answer")
    print("True")
else:
    print("Answer")
    print("False")
```

- Enquanto o seguinte código gera erro

```
if True:
    print("Answer")
    print("True")
else:
    print("Answer")
  print("False")
```

Instruções de Múltiplas Linhas e Comentários

- Pode-se usar o caractere "\" para utilizar múltiplas linhas numa instrução

```
x = 10
y = 1.1
total = x + \
        y + \
        3.14
dias = ['Segunda', 'Terça', 'Quarta',
        'Quinta', 'Sexta']
```

- Python aceita aspas simples (’), duplas (") e triplas (’’’ ou """)

```
palavra = 'palavra'
sentenca = "Essa é uma sentença."
paragrafo = """Este é um parágrafo. Ele é
composto por múltiplas linhas."""
```

- Um símbolo de hash (#) que não esteja dentro de uma *string* inicia um comentário

Múltiplas Instruções em uma Linha

- Pode-se escrever múltiplas instruções em uma linha

```
import sys; x = 'palavra'; sys.stdout.write(x + '\n')
```

Tipos de Dados Básicos

```
contador     = 100          # Valor inteiro
distancia    = 1000.0       # Valor em ponto flutuante
nome         = "Pedro"      # Texto

lista        = [ 'abcd', 786 , 2.23, 'john', 70.2 ] # Lista
print(lista) # ['abcd', 786, 2.23, 'john', 70.2]
print(lista[2]) # 2.23
print(lista[1:3]) # [786, 2.23]
print(lista[2:]) # [2.23, 'john', 70.2]
print(lista[:2]) # ['abcd', 786]

tupla        = ( 'abcd', 786 , 2.23, 'john', 70.2  ) # Tupla; leitura
conjunto     = { 1, 2, 10, 1, 2 } # Conjunto
print(conjunto) # {1, 2, 10}
```

Tipos de Dados Básicos

```
dicionario  = {}              # Dicionário
dicionario['um'] = "Primeiro elemento"
dicionario[2]    = "Segundo elemento"
dic = {'nome': 'Pedro','Matrícula':123, 'Curso': 'Algum...'}
print( dic.keys() ) # Lista de chaves do dicionário
# dict_keys(['Curso', 'Matrícula', 'nome'])
print( dic.values() ) # Lista de valores do dicionário
# dict_values(['Algum...', 123, 'Pedro'])
print( dic[ "nome" ] )
# Pedro
```

Atribuição de Valores e Desalocação de Memória

- Atribuições

```
contador     = 100
a = b = c = 1
a,b,c = 1,2,"Pedro"
print(a)
del a
del b,c
```

- Conversão de valores

```
inteiro = int("123")
inteiro_de_binario = int("1010", 2)
real = float("3.14")
texto = "pi = " + str(real)
```

- Outras funções para conversão de tipos:

  http://www.tutorialspoint.com/python/python_variable_types.htm

Operações Aritméticas

Operator	Description	Example
+ Addition	Adds values on either side of the operator.	a + b = 30
- Subtraction	Subtracts right hand operand from left hand operand.	a – b = -10
* Multiplication	Multiplies values on either side of the operator	a * b = 200
/ Division	Divides left hand operand by right hand operand	b / a = 2
% Modulus	Divides left hand operand by right hand operand and returns remainder	b % a = 0
** Exponent	Performs exponential (power) calculation on operators	a**b =10 to the power 20
//	Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed. But if one of the operands is negative, the result is floored, i.e., rounded away from zero (towards negative infinity):	9//2 = 4 and 9.0//2.0 = 4.0, -11//3 = -4, -11.0//3 = -4.0

Operações de Comparação

Operator	Description	Example
==	If the values of two operands are equal, then the condition becomes true.	(a == b) is not true.
!=	If values of two operands are not equal, then condition becomes true.	
>	If the value of left operand is greater than the value of right operand, then condition becomes true.	(a > b) is not true.
<	If the value of left operand is less than the value of right operand, then condition becomes true.	(a < b) is true.
>=	If the value of left operand is greater than or equal to the value of right operand, then condition becomes true.	(a >= b) is not true.
<=	If the value of left operand is less than or equal to the value of right operand, then condition becomes true.	(a <= b) is true.

Outras Operações

- Conjuntos

```
tupla = ( 'abcd', 786 , 2.23, 'john', 70.2  )
x = 2.23
print( x in tupla )

x = "786"
print( x in tupla )
```

- Identificadores

```
x = "786"
y = str(786)

print( x == y )
print( x is y )
```

Condicionais

```
tupla = ( 'abcd', 786 , 2.23, 'john', 70.2  )
x = "786"

if x in tupla:
        print(str(x) + " está na tupla.")
elif ( "70.2" in tupla or 2.23 in tupla ):
        print("2.23 está na tupla.")
else:
        print(str(x) + " não está na tupla.")
```

```
2.23 está na tupla.
```

Estruturas de Repetição

```
c = 0
while (c < 9):
    print('Contagem:', c)
    c += 1

print("Fim")
```

```
Contagem:  0
Contagem:  1
Contagem:  2
Contagem:  3
Contagem:  4
Contagem:  5
Contagem:  6
Contagem:  7
Contagem:  8
Fim
```

Estruturas de Repetição

```
c = 0
while c < 3:
    print(c, " é menor do que 3")
    c = c + 1
else:
    print(c, " não é menor do que 3")
```

```
0 é menor do que 3
1 é menor do que 3
2 é menor do que 3
3 não é menor do que 3
```

Estruturas de Repetição

```
for letra in 'Python':
    print(letra)

tupla = ( 'abcd', 786 , 2.23, 'john', 70.2  )
for t in tupla:
    print(t)
```

```
P
y
t
h
o
n
abcd
786
2.23
john
70.2
```

Estruturas de Repetição

```
for i in range(1,5):
    for j in range(0,3):
        print(i,"x",j,"=",i*j)
```

```
1 x 0 = 0
1 x 1 = 1
1 x 2 = 2
2 x 0 = 0
2 x 1 = 2
2 x 2 = 4
3 x 0 = 0
3 x 1 = 3
3 x 2 = 6
4 x 0 = 0
4 x 1 = 4
4 x 2 = 8
```

Funções Matemáticas e Bibliotecas

```
import math

x = 4.0
print(math.exp(x))
print(math.sqrt(x))
print(math.e)
print(math.pi)

from math import exp

print(exp(x))
```

```
54.598150033144236
2.0
2.718281828459045
3.141592653589793
54.598150033144236
```

http://www.tutorialspoint.com/python/python_numbers.htm

Strings

- *Strings* podem ser manipuladas como listas

```
str1 = 'Olá Mundo!'
str2 = "Python"

print("str1[0]: ", str1[0])
print("str2[1:5]: ", str2[1:5])
print( str1[:3] + " " + str2 )
```

```
str1[0]:  O
str2[1:5]:  ytho
Olá Python
```

- Formatação

```
print("Meu nome é %s e minha idade é %d anos." % ('Pedro', 23) )
# Meu nome é Pedro e minha idade é 23 anos.
```

Strings

- Algumas funções para *Strings* podem ser encontradas em https://www.tutorialspoint.com/python/python_strings.htm

```
str = 'Uma string para fazer um teste'

for s in str.split():
        print(s)
```

```
Uma
string
para
fazer
um
teste
```

Definição de Funções

```
def nome_da_funcao( parametros ):
    "texto para documentar a função"
    intrucoes_da_funcao
    return [ expressao ]
```

Definição de Funções

- Exemplo de como calcular uma exponencial para valores próximos de zero

```
def exponencial( x ):
    "calcula uma aproximação para a exponencial"
    resultado = 0.0
    denominador = 1.0
    numerador = 1.0
    for i in range(1, 11):
        resultado += numerador / denominador
        numerador *= x
        denominador *= i
    return resultado

print( exponencial( 1 ) )
# 2.7182818011463845
print( exponencial( 2 ) )
# 7.388994708994708
```

Definição de Funções

- Exemplo com retorno múltiplo

```
import math
def funcao( x ):
    "calcula o quadrado e a raiz do parâmetro"
    return x*x, math.sqrt(x)

p, r = funcao( 4 )
print( p )
# 16
print( r )
# 2.0
s = funcao( 2 )
print( s )
# (4, 1.4142135623730951)
print( type( s ) )
# <class 'tuple'>
```

Definição de Funções

- Parâmetros são passados por referência

```
def funcao( lista ):
   lista.append( 1 )
   lista.append( 2 )
   lista.append( 3 )
   print("Valores na lista: ", lista)
   # Valores na lista:  [10, 20, 30, 1, 2, 3]

lista = [10,20,30];
funcao( lista );
print("Valores na lista: ", lista)
# Valores na lista:  [10, 20, 30, 1, 2, 3]
```

Definição de Funções

- Valores para as variáveis e valores padrões

```
def exponencial( x, n=10 ):
    "calcula uma aproximação para a exponencial"
    resultado = 1.0
    denominador = 1.0
    numerador = 1.0
    for i in range(1, n+1):
        numerador *= x
        denominador *= i
        resultado += numerador / denominador
    return resultado

print( exponencial( n=10, x=1 ) )
# 2.718281801146384 5
print( exponencial( 1.0 ) )
# 2.718281801146384 5
```

- Existem outras características nas definições de funções

Exercício 1

- Crie uma função para gerar uma sequência de Fibonacci
- A função deve ter um parâmetro: número de termos da sequência
- O argumento deve assumir um valor padrão igual a 10
- A função deve retornar uma tupla com os termos da sequência
- Teste a função criada

Entrada e Saída de Dados

- Leitura de uma linha de texto

```
str = input("Digite seu texto: ");
print("Texto digitado: ", str)
```

Entrada e Saída de Dados

- Escrevendo arquivos de texto

```
arq = open("arq.txt", "w") # abre arquivo para escrita
print("Nome: ", arq.name) # nome do arquivo
print("Fechado: ", arq.closed) # verifica se o arquivo está fechado
print("Modo: ", arq.mode) # modo em que o arquivo foi aberto
arq.write("texto1\n") # escreve no arquivo
arq.write("texto2\n")
arq.write("texto3")
arq.close() # fecha o arquivo
```

Entrada e Saída de Dados

- Lendo arquivos de texto

```
arq = open("arq.txt", "r") # abre arquivo para leitura

str = arq.read() # leitura do conteúdo do arquivo
arq.close() # fecha o arquivo

for s in str.split("\n"): # percorre cada linha do arquivo
        print( s ) # imprime a linha
```

- Python possibilita outras manipulações em arquivos

Entrada e Saída de Dados: Colab

```
# Import PyDrive and associated libraries.
# This only needs to be done once per notebook.
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials

# Authenticate and create the PyDrive client.
# This only needs to be done once per notebook.
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)

# Download a file based on its file ID.
file_id = '1ZQIHL8aU0Rj3lp3h8MtPz4KLn4DRR5io'
downloaded = drive.CreateFile({'id': file_id})
print('Downloaded content "{}"'.format(downloaded.GetContentString()))
```

Entrada e Saída de Dados: Colab

```
# Escreve em arquivo criado à partir da raíz do Drive
meuArquivo = drive.CreateFile({'title': 'arquivo-de-teste.txt'})
meuArquivo.SetContentString('teste1\nteste2')
meuArquivo.Upload() # Upload file.

# Lê arquivo e insere um texto no final dele
meuArquivo = drive.CreateFile({'id':'1CDfoh4wsWc_UgyvWwRCkosj4jb8LSgJD'})

texto = meuArquivo.GetContentString()
texto += "\nnovo texto inserido no arquivo"
meuArquivo.SetContentString(texto)

meuArquivo.Upload() # Upload file.
```

Entrada e Saída de Dados: Colab

- Pode-se montar a estrutura de arquivos do Google Drive
- Permite acessar os arquivos via caminho (em relação à raiz) e nome do arquivo

```
# importa biblioteca
from google.colab import drive

# monta a estrutura de arquivos do Google Drive
drive.mount('/content/drive')

# abre o arquivo pra leitura e guarda os dados numa variável
arq = open("/content/drive/MyDrive/Classroom/.../reg_2_tr_X.dat", "r")
str = arq.read() # leitura do conteúdo do arquivo
arq.close() # fecha o arquivo

# quebra o texto do arquivo entre as linhas e
# imprime o conteúdo de cada linha
for s in str.split("\n"): # percorre cada linha do arquivo
        print( s ) # imprime a linha
```

Exceções

- Pode-se tratar exceções

```
try:
    print(2/0)
except ZeroDivisionError:
    print("Divisão por zero!")
finally:
    print("Fim do teste")

try:
        a = [ 1, 2, 3 ]
        print( a[10] )
except IndexError:
    print("Índice inválido!")
```

```
Divisão por zero!
Fim do teste
Índice inválido!
```

Gráficos

```
from numpy import *
import matplotlib.pyplot as plt

x = array([1,2,3,4,5]) # gera um array com os elementos
y = x**2 + 1 # operações vetoriais sobre x

print( x ) # imprime x
print( y ) # imprime y

# determina os coeficientes de polinômio de grau 2 via mínimos quadrados
pol = polyfit(x,y,2)
p1 = poly1d( pol ) # encapsula o polinômio
p2 = poly1d(polyfit(x,y,1))

print( pol )
print( p1 )
print( p2 )
```

Gráficos – continuação

```
y1 = p1(x) # avalia x no polinômio
y2 = p2(x)

f = plt.figure() # gera uma figura do gráfico (antes de desenhá-lo)

plt.plot(x, y, "bo", x, y1, x, y2) # gera o gráfico
plt.show() # exibe o gráfico

f.savefig("fig.pdf") # gera um arquivo pdf com o gráfico
```

Exercício 2

- Gere um gráfico e salve o arquivo em PDF
- O gráfico deve conter os pontos presentes nos arquivos
  `reg_2_tr_X.dat`
  `reg_2_tr_Y.dat`
- O gráfico deve conter a reta que melhor se ajusta aos pontos segundo o critério dos mínimos quadrados
- Crie uma função para ler os arquivos
  `leArquivo(nomeArq, strPular )`
- Extra: Gere o gráfico com um polinômio de grau 6
- Use a função `sort` para ordenar os elementos do *array* `x`

Exercício 2

- Gere um gráfico e salve o arquivo em PDF
- O gráfico deve conter os pontos presentes nos arquivos
  `reg_2_tr_X.dat`
  `reg_2_tr_Y.dat`
- O gráfico deve conter a reta que melhor se ajusta aos pontos segundo o critério dos mínimos quadrados
- Crie uma função para ler os arquivos
  `leArquivo(nomeArq, strPular )`
- Extra: Gere o gráfico com um polinômio de grau 6
- Use a função `sort` para ordenar os elementos do *array* `x`