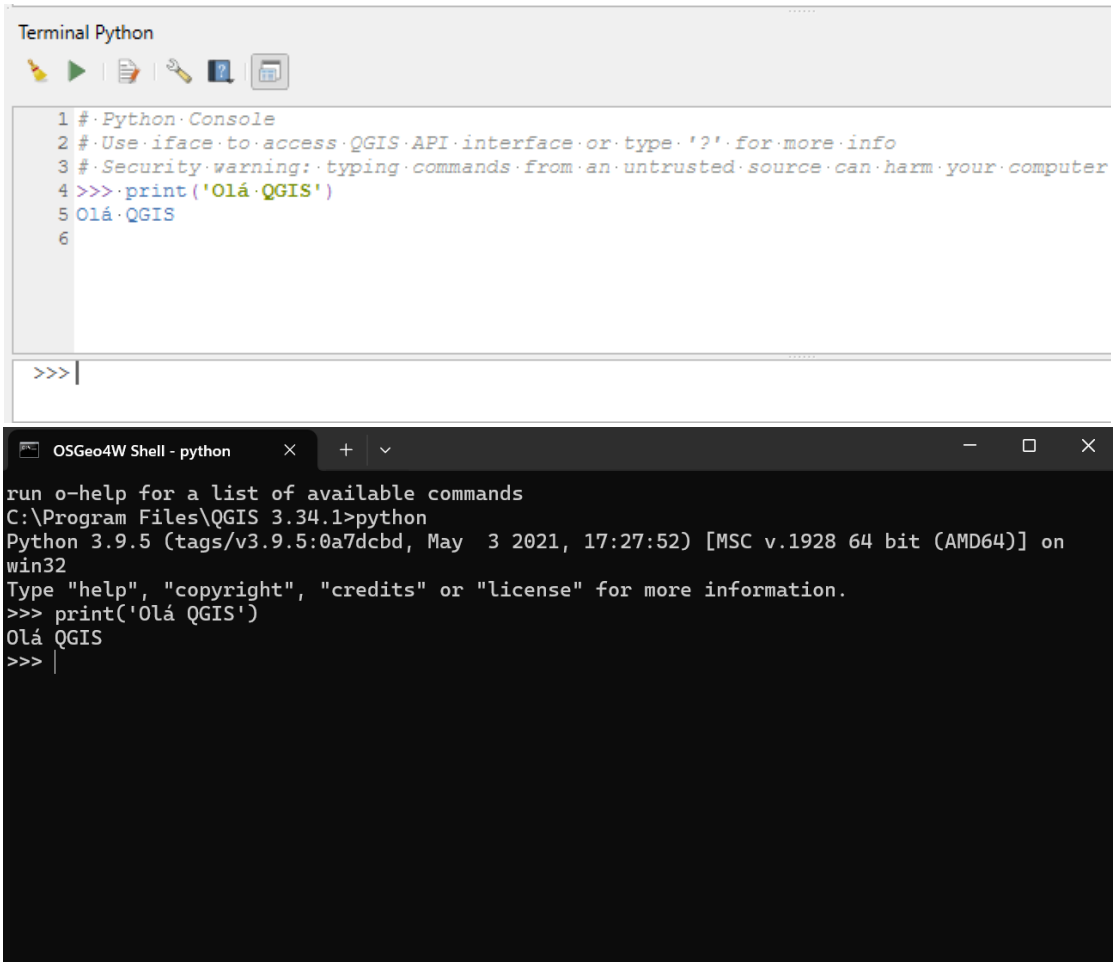
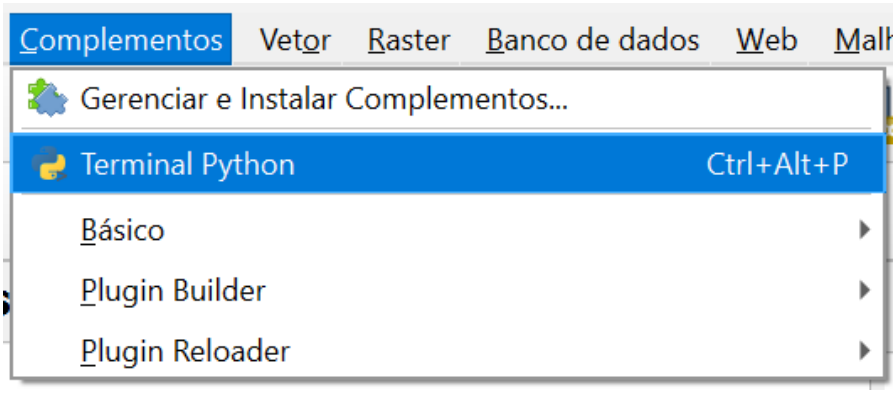


Aprendendo Python no QGIS

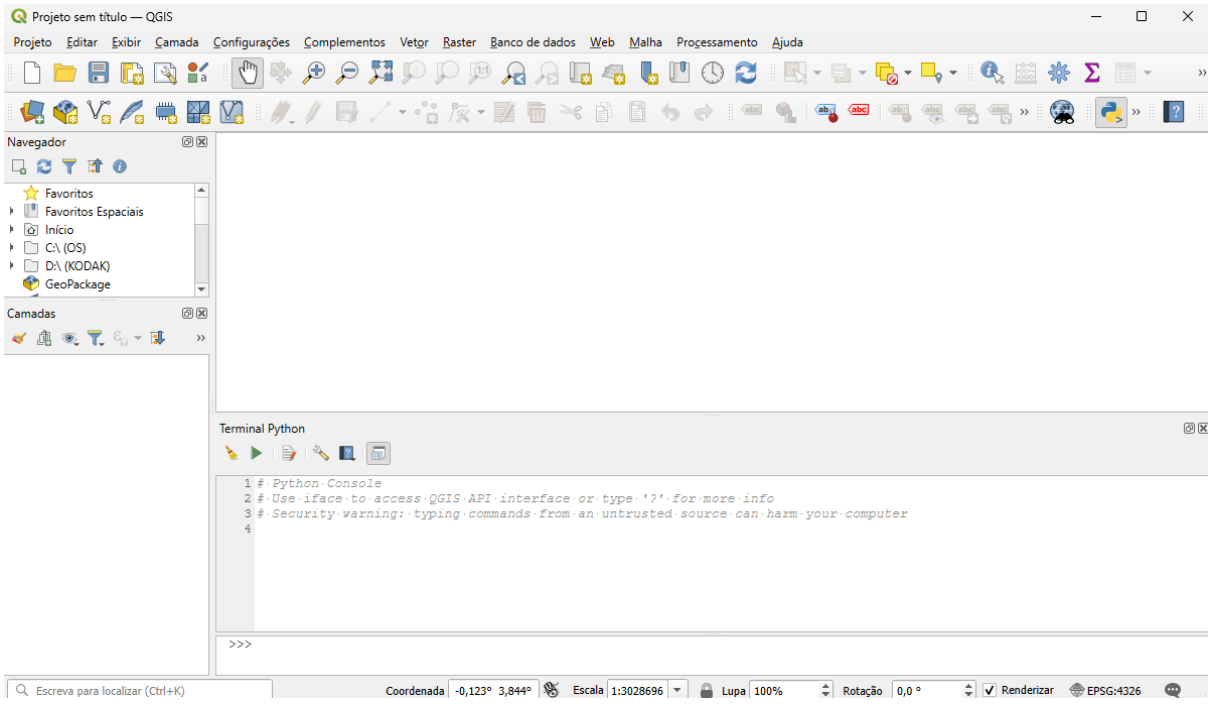
Usando o Terminal Python ou o Terminal OSGeo4W

1 – O Terminal Python

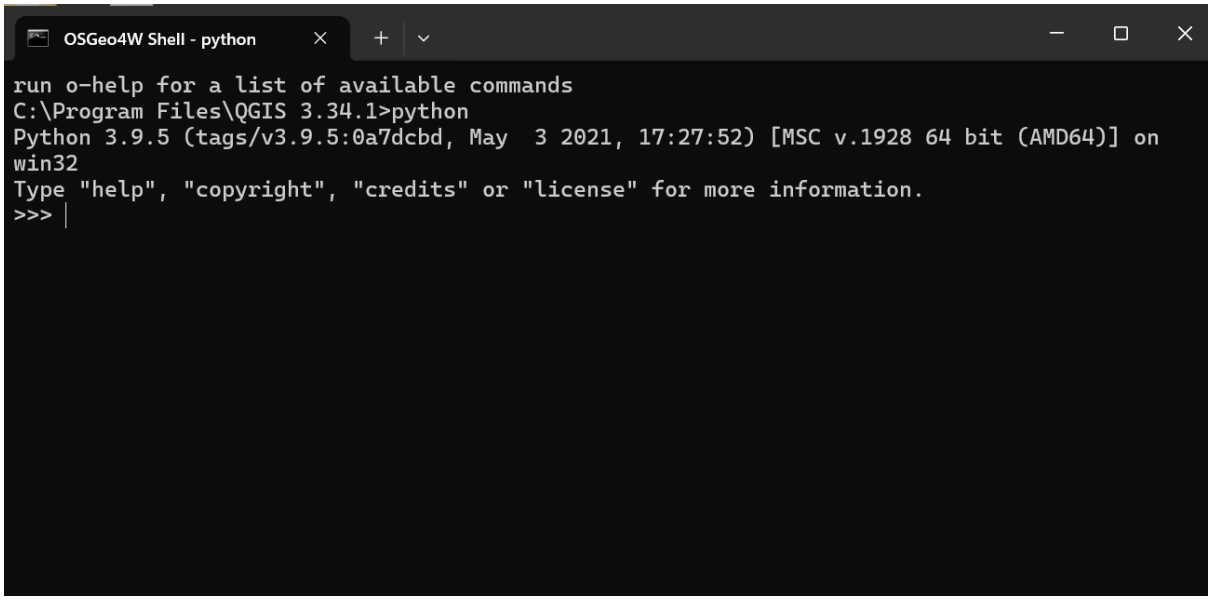
Para iniciar o terminal python no QGIS vá em Complementos→Termina python ou pressione **Ctrl-Alt-P**.



Veremos na parte inferior da tela o Terminal python aberto.



Esse é um tutorial de introdução à linguagem python que usa o terminal python do QGIS como nosso ambiente de aprendizado. Caso prefira, pode também utilizar o terminal diretamente via programa OSGeo4W Shell e digitando python para inicializar o terminal.



2 - Fundamentos da linguagem Python

Comentários de script iniciam com # em python.

```
>>> # um comentário
>>> print('QGIS') # outro comentário
QGIS
>>> a ='# isso não é um comentário pois está entre aspas'
```

Tipos Numéricos

Python possui primitivamente os tipos numéricos de números inteiros (int), ponto flutuante (float) e complexo (complex).

```
>>> a = 1
>>> b = 3.2
>>> c = 4 + 3j
```

A função type(variável) retorna o tipo da variável.

```
>>> type(a)
<class 'int'>
>>> type(b)
<class 'float'>
>>> type(c)
<class 'complex'>
```

O resultado de uma divisão sempre será do tipo float.

```
>>> d = a/a
>>> d
1.0
>>> type(d)
<class 'float'>
```

O console pode funcionar como uma calculadora também.

```
>>> 2+2
4
>>> 2**8 #dois elevado a oito
256
>>> 82%3 #resto inteiro da divisão
1
>>> 82//3 #quociente da divisão
27
>>> 3-2
1
>>> (50-5*6)/4 # Parênteses tem precedente na operação
5.0
```

Tipo Alfanumérico

Em python o tipo str é usado para palavras, textos e caracteres alfanuméricos.

```
>>> nome = 'Manuel'
>>> type(nome)
<class 'str'>
>>> letra = 'a'
>>> type(letra)
<class 'str'>
```

Um objeto da classe str nada mais é do que uma matriz de valores sequenciados onde o

primeiro valor (caractere) corresponde ao índice 0 da matriz e o último valor ao índice -1, o penúltimo -2 e assim sucessivamente.

```
>>> nome[0]
'M'
>>> nome[1]
'a'
>>> nome[-1]
'l'
>>> nome[-2]
'e'
>>> nome[-6]
'M'
>>> nome[2:5] #note que o valor do índice 5 não está incluído
'nue'
```

Um objeto str uma vez definido é imutável e se tentarmos mudar o valor de um objeto str uma mensagem de erro aparecerá.

```
>>> nome[2]='t'
Traceback (most recent call last):
File "/usr/lib/python3.7/code.py", line 90, in runcode
exec(code, self.locals)
File "<input>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

A função len() retorna o comprimento do objeto str.

```
>>> palavra='anticonstitucionalissimamente'
>>> len(palavra)
29
```

Podemos usar aspas simples ou duplas para delimitar um objeto str. Isso é prático para para podermos definir strings com estas:

```
>>> s1 = "Bom dia senhor O'Brien"
>>> print(s1)
Bom dia senhor O'Brien
>>> s2= 'Quoth the raven "Nevermore". '
>>> print(s2)
Quoth the raven "Nevermore".
```

Tipo lista

Python possui diversos tipos compostos de dados, a lista (list) é uma delas.
Uma lista é um conjunto de objetos de determinado tipo ou de tipos distintos armazenados em uma lista.
Listas são declaradas dentro de colchetes onde cada objeto (item) dela é separado por vírgula.

```
>>> lista = [3200,2670,3100,3000]
>>> caipi = ['gelo', 'limão','açúcar',51,15.99]
```

Assim como str, cada item de uma lista pode ser acessado usando índices.

```
>>> lista[0]
3200
>>> lista[1]
2670
>>> lista[3]
```

```
3000
>>> lista[1:3]
[2670, 3100]
>>> type(caipi [0])
<class 'str'>
>>> type(caipi [3])
<class 'int'>
>>> type(caipi [4])
<class 'float'>
>>> caipi [-1]
15.99
```
Ao contrário do objeto str, os valores de itens de uma lista podem ser modificados
```
>>> caipi[3]='pinga'
>>> caipi
['gelo', 'limão', 'açúcar', 'pinga', 15.99]
```
Uma lista pode conter outras listas como itens. E acessamos cada item dessa lista interna usando um índice adicional.
```
>>> caipi[1]=['abacaxi','maracujá','limão']
>>> caipi
['gelo', ['abacaxi', 'maracujá', 'limão'], 'açúcar', 'pinga', 15.99]
>>> caipi[1][1]
'maracujá'
>>> caipi[1][-1]
'limão'
```
Podemos adicionar itens a uma lista já existente usando adição ou a função append().
```
>>> caipi = caipi + ['guardanapo','canudo']
>>> caipi.append('copo')
>>> caipi
['gelo', ['abacaxi', 'maracujá', 'limão'], 'açúcar', 'pinga', 15.99,'guardanapo','canudo', 'copo']
```
Alguns métodos de interação com listas são mostrados abaixo:
Criamos a seguinte lista vazia inicialmente.
```
>>> lis=[]
>>> lis
[]
```
Adicionando itens na lista com o método extend().
```
>>> lis.extend([1,2,3])
>>> lis
[1, 2, 3]
```
Adicionando um item de valor 10 na posição predeterminada (0) com o método insert().
```
>>> lis.insert(0,10)
>>> lis
[10, 1, 2, 3]
```
Removendo da lista a primeira ocorrência do valor passado pelo método remove().
```
>>> lis.remove(2)
>>> lis
[10, 1, 3]
```
Podemos copiar uma lista usando o método copy().
```
>>> lis2=lis.copy()
>>> lis2
[10, 1, 3]
```

Revertemos a ordem dos itens de uma lista com o método reverse().

```
>>> lis2.reverse()
>>> lis2
[3, 1, 10]
```

Ordenamos uma lista usando o método sort().

```
>>> lis2.sort()
>>> lis2
[1, 3, 10]
```

O método pop() remove e retorna o item indicado pelo índice dado, se nenhum índice é fornecido o último item é removido da lista e retornado.

```
>>> lis3 = [1,1,2,3,4,4,4,3,2,3,1]
>>> lis3.pop(1)
1
>>> lis3
[1, 2, 3, 4, 4, 4, 3, 2, 3, 1]
```

O método index(x[,início[,fim]]) retorna o índice (na base 0) da primeira ocorrência do item x. O segundo argumento mostra a partir de qual e até qual índice da lista procurar. Caso não encontre um item com o valor uma mensagem de erro é retornada.

```
>>> lis3.index(1,1,10)
9
>>> lis3.index(1)
0
>>> lis3.index(1,1)
9
>>> lis3.index(1,1,10)
9
>>> lis3.index(8,1,10)
Traceback (most recent call last):
File "C:/PROGRA~1/QGIS3~1.4/apps/Python37/lib/code.py", line 90, in
runcode
exec(code, self.locals)
File "<input>", line 1, in <module>
ValueError: 8 is not in list
```

O método count(x) retorna o número de vezes que o item x aparece na lista.

```
>>> lis3.count(4)
3
```

O método clear() remove todos os itens da lista.

```
>>> lis3.clear()
>>> lis3
[]
```

Podemos usar a instrução del para deletar itens de uma lista usando índices em vez de valores.

```
>>> lis3 = [1,1,2,3,4,4,4,3,2,3,1]
>>> del lis3[2]
>>> lis3
[1, 1, 3, 4, 4, 4, 3, 2, 3, 1]
>>> del lis3[5:8]
>>> lis3
[1, 1, 3, 4, 4, 3, 1]
```

Tuples

Assim como str e listas, tuples são dados em sequência usados em python. As diferenças principais entre uma tuple e uma lista é que tuples são declaradas usando vírgulas para separar os itens e esses itens são imutáveis.

```
>>> t = 'banana', 3,45, False, "oi!"
>>> t
('banana', 3, 45, False, 'oi!')
>>> t[0]
'banana'
>>> doist =t,(1,2,3,4,5)
>>> doist
(('banana', 3, 45, False, 'oi!'), (1, 2, 3, 4, 5))
>>> doist[0]
('banana', 3, 45, False, 'oi!')
>>> doist[0][0]
'banana'
>>> doist[1][0]
1
>>> t[0]='maçã'
Traceback (most recent call last):
File "C:/PROGRA~1/QGIS3~1.4/apps/Python37/lib/code.py", line 90, in
runcode
exec(code, self.locals)
File "<input>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

Sets

Sets ou conjuntos são outro tipo de dados em sequência que python utiliza. Sets são definidos dentro de chaves {} e resultam em uma simples aparição de cada um de seus itens e estes não são indexados.

```
>>> conjunto = {'rio','terra','fogo','fogo','água','terra'}
>>> conjunto
{'terra', 'fogo', 'rio', 'água'}
>>> 'rio' in conjunto
True
>>> 'mar' in conjunto
False
>>> num ={1,2,3,2,3,2,1,23,'a'}
>>> num
{1, 2, 3, 'a', 23}
>>> cidade=set('Pindamonhongaba')
>>> cidade
{'p', 'a', 'h', 'm', 'g', 'n', 'i', 'o', 'b', 'd'}\
```

Dicionários

Dicionários são um tipo de dados bastante usado em python. Um dicionário possui sempre uma chave e um valor, esta chave é usada no lugar de um índice numérico que

é usado numa lista. Essa chave deve ser única e um dicionário vazio pode ser criado usando {}.

```
>>> dicio={'nome':'andre','sobrenome':'costa'}
>>> dicio['idade']=56
>>> dicio
{'nome': 'andre', 'sobrenome': 'costa', 'idade': 56}
>>> list(dicio)
['nome', 'sobrenome', 'idade']
>>> sorted(dicio)
['idade', 'nome', 'sobrenome']
>>> dicio[1]
Traceback (most recent call last):
File "C:/PROGRA~1/QGIS3~1.4/apps/Python37/lib/code.py", line 90, in
runcode
exec(code, self.locals)
File "<input>", line 1, in <module>
KeyError: 1
>>> del dicio['idade']
>>> dicio
{'nome': 'andre', 'sobrenome': 'costa'}
>>> dicio2=dict([('código', 34), ('senha', 65483), ('acessos', 8)])
>>> dicio2
{'código': 34, 'senha': 65483, 'acessos': 8}
```

Funções Internas

As seguintes funções internas são usadas para conversão de tipos e informações de tipos.

```
>>> f=-5.789
>>> round(f,2)
-5.79
>>> abs(f)
5.789
>>> int(f)
-5
>>> str(f)
'-5.789'
>>> lis=[2,3,45,12,78,1,-17,3]
>>> min(lis)
-17
>>> max(lis)
78
>>> sum(lis)
127
>>> sorted(lis)
[-17, 1, 2, 3, 3, 12, 45, 78]
>>> i=255
>>> hex(i)
'0xff'
>>> bin(i)
'0b11111111'
```

```
>>> float(i)
255.0
>>> chr(i)
'ÿ'
>>> ord('ÿ')
255
>>> oct(i)
'0o377'
>>> ascii(i)
'255'
>>> pow(2,10)
1024
>>> bool(1<2 and 3>2)
True
>>> bool(1<2 and 3>4)
False
```

As seguintes palavras são reservadas da linguagem python e não podem ser usadas para definir variáveis.

and	**except**	**lambda**	**with**
as	**finally**	**nonlocal**	**while**
assert	**False**	**None**	**yield**
break	**for**	**not**	
class	**from**	**or**	
continue	**global**	**pass**	
def	**if**	**raise**	
del	**import**	**return**	
elif	**in**	**True**	
else	**is**	**try**	

E essas são as funções internas:

abs()	**delattr()**	**hash()**	**memoryview()**	**set()**
all()	**dict()**	**help()**	**min()**	**setattr()**
any()	**dir()**	**hex()**	**next()**	**slice()**
ascii()	**divmod()**	**id()**	**object()**	**sorted()**
bin()	**enumerate()**	**input()**	**oct()**	**staticmethod()**
bool()	**eval()**	**int()**	**open()**	**str()**
breakpoint()	**exec()**	**isinstance()**	**ord()**	**sum()**
bytearray()	**filter()**	**issubclass()**	**pow()**	**super()**
bytes()	**float()**	**iter()**	**print()**	**tuple()**
callable()	**format()**	**len()**	**property()**	**type()**
chr()	**frozenset()**	**list()**	**range()**	**vars()**
classmethod()	**getattr()**	**locals()**	**repr()**	**zip()**
compile()	**globals()**	**() reversed()**	**__import__()**	
complex()	**hasattr()**	**max()**	**round()**	

2 - Controles de fluxo

Como toda linguagem de programação, python utiliza controles de fluxo de programa. Mas antes disso vamos falar um pouco de recuo de script. Obrigatoriamente em python temos de usar recuo de blocos de código uma vez que não os separamos por chaves {}, assim todo bloco de código, seja ele uma classe, função ou um controle de fluxo, deve ser indentado. Num bloco com recuo no console, >>> se transforma em ... indicando que estamos dentro de um bloco no script. Quando todas as instruções do bloco estão finalizadas, teclamos ‘enter’ na linha final com ...

if elif else

O if talvez seja a instrução mais conhecida em programação. Ela checa se (if) uma condição ou se outras condições (elif) são atendidas. Se nenhuma condição for atendida, podemos também instruir que algo seja feito (else).

```
>>> x = int(input("Diga um número inteiro: "))
```

Diga um número inteiro: 2

```
>>> if x<0:
... print('O número é negativo')
... elif x>0:
... print('O número é positivo')
... else:
... print('O número é zero')
...
```

O número é positivo

for

A repetição (loop) for é definida como “executar/repetir as instruções nos termos predefinidos”. No exemplo abaixo a repetição é feita para cada palavra da variável palavras e a palavra e o comprimento dela é impresso como resultado.

```
>>> palavras = ['Olá!','Vamos','aprender','python?']
>>> for palavra in palavras:
... print(palavra,len(palavra))
...
```

Olá! 4
Vamos 5
aprender 8
python? 7

while

A repetição while é definida com “enquanto o que foi predefinido não ocorrer, vai executando/repetindo”.

```
>>> a, b = 0, 1
>>> while a < 1000:
... print(a, end=',')
... a, b = b, a+b
...
```

0,1,1,2,3,5,8,13,21,34,55,89,144,233,377,610,987,

range()

A forma que python interage com uma série numérica é usando a função range().

```
>>> for i in range(10):
... print(i,end=',')
...
```

0,1,2,3,4,5,6,7,8,9,

Vamos supor que não sabemos a quantidade de itens numa lista mas queremos interagir (no caso listar) cada um destes itens. Usamos range para nos auxiliar.

```
>>> palavras = ['Olá!','Vamos','aprender','python?']
>>> for i in range(len(palavras)):
... print(palavras[i],i)
...
Olá! 0
Vamos 1
aprender 2
python? 3
```

Podemos definir o início e final de range() e o passo também.

```
>>> list(range(50,57))
[50, 51, 52, 53, 54, 55, 56]
>>> list(range(10,5,-1))
[10, 9, 8, 7, 6]
>>> list(range(-100,-50,10))
[-100, -90, -80, -70, -60]
```

Break, continue e else em loops

A instrução break quebra a execução da repetição for ou while mais interna.
Repetições também podem ter uma instrução else; ela é executada quando a repetição termina mas não quando ela é interrompida por uma instrução break. Veja o exemplo abaixo:

```
>>> for n in range(2,10):
... for x in range(2, n):
... if n % x == 0:
... print(n, 'é igual a ', x, '*', n//x)
... break
... else:
... print(n, 'é um número primo')
...
2 é um número primo
3 é um número primo
4 é igual a 2 * 2
5 é um número primo
6 é igual a 2 * 3
7 é um número primo
8 é igual a 2 * 4
9 é igual a 3 * 3
```

A instrução continue, avança para a próxima interação da repetição:

```
>>> for num in range(2, 6):
... if num % 2 == 0:
... print("Número par", num)
... continue
... print("Número ímpar", num)
...
Número par 2
Número ímpar 3
Número par 4
Número ímpar 5
```

3 - Funções

Uma função é um conjunto de instruções organizadas e reusáveis para executar uma tarefa. Em python definimos uma função usando def seguido do nome da função e dos argumentos ou parâmetros passados dentro de parênteses. Veja o exemplo abaixo:

```
>>> def fibo(n):
... """Calcula a série de Fibonacci até o limite informado"""
... a,b=0,1
... while a<n:
... print(a,end=' ')
... a,b=b,a+b
... print()
...
>>> fibo(100)
0 1 1 2 3 5 8 13 21 34 55 89
>>> fibo(1000)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
>>> fibo(10000)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765
```

Uma outra forma de escrever essa função, utilizando retornando no formato de lista, é mostrada a seguir:

```
>>> def fibo2(n):
... """Calcula e retorna a série de Fibonacci até o limite
informado"""
... resultado=[]
... a,b=0,1
... while a<n:
... resultado.append(a)
... a,b=b,a+b
... return resultado
...
>>> fibo2(100)
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
```

Podemos assinalar uma função a uma variável.

```
>>> f=fibo2
>>> f(10)
[0, 1, 1, 2, 3, 5, 8]
```

Se passarmos nossa função como argumento para a função help teremos a string de documentação (entre as aspas duplas repetidas três vezes) como informação retornada. Essa é uma maneira de documentar o que uma função faz em python.

```
>>> help(fibo2)
Help on function fibo2 in module __main__:
fibo2(n)
Calcula e retorna a série de Fibonacci até o limite informado
```

É possível definir funções com número de argumentos variáveis.

Valor padrão predefinido é a forma mais comum onde podemos predefinir o valor de um ou mais parâmetros de uma função.

```
>>> def pergunta(texto, tentativas=4, msg='Tente de novo!'):
... while True:
... ok = input(texto)
... if ok in ('Sim', 'sim', 's', 'S'):
```

```
... return True
... if ok in ('n', 'não', 'N', 'Não', 'nao', 'Nao'):
... return False
... tentativas = tentativas - 1
... if tentativas < 0:
... raise ValueError('Resposta Inválida.')
... print(msg)
...
>>> pergunta('Está com fome?')
Está com fome?j
Tente de novo!
Esta com fome?Sim
True
>>> pergunta('Está com fome?',1,'Não entendi a resposta!')
Está com fome?iop
Não entendi a resposta!
Está com fome?poi
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
File "<stdin>", line 10, in pergunta
ValueError: Resposta Inválida.
```

Vamos ver agora um exemplo prático de como trabalhar com funções criando um nosso primeiro script. Crie o arquivo **temp_converter.py** com o seguinte código.

```
#!/usr/bin/env python3
'''Converte temperaturas de Celsius para Fahrenheit, Kelvin para
Celsius e Kelvin
para Fahrenheit.
Uso:
Carregue usando import
import temp_converter as tc
Autor:
Seu Nome - 03.12.2019'''
def celsius_para_fahr(temp_celsius):
   return 9/5 * temp_celsius + 32
def kelvins_para_celsius(temp_kelvins):
   return temp_kelvins - 273.15
def kelvins_para_fahr(temp_kelvins):
   temp_celsius = kelvins_para_celsius(temp_kelvins)
   temp_fahr = celsius_para_fahr(temp_celsius)
   return temp_fahr
```

Agora vamos importar nosso script e usar funções dele.

```
>>> import temp_converter as tc
>>> print("O ponto de congelamento da água em Fahrenheint é:",
tc.celsius_para_fahrenheint(0))
O ponto de congelamento da água em Fahrenheit é: 32.0
>>> print('O Zero absoluto em Fahrenheit é:',
tc.kelvins_para_fahr(0))
O Zero absoluto em Fahrenheit é: -459.66999999999996
```

4 - Módulos

Em python um module (módulo) é simplesmente um arquivo com extensão .py com classes, funções e demais instruções. Nosso script acima é um exemplo de um módulo. Um package (pacote) é uma forma de organizar vários módulos em uma entidade maior. Algumas linguagens chamam módulos e pacotes de library. Um módulo é carregado usando o comando **`import`** e podemos renomear um módulo usando **`as`**

```
>>> import math as m
>>> m.sqrt(81)
9
```

Também podemos importar uma simples função de um módulo usando from

```
>>> from math import sqrt
>>> sqrt(9)
3
```

Ou podemos importar um submódulo de um módulo.

```
>>> import matplotlib.pyplot as plt
>>> plt.figure()
<Figure size 432x288 with 0 Axes>
```

Usaremos o tempo todo vários módulos, essa é e força da linguagem python com inúmeros módulos existentes para as mais diversas funções. Vamos iniciar vendo os módulos pandas e matplotlib.

5 Pandas

A biblioteca pandas foi desenvolvida como uma alternativa a linguagem R para lidar com estrutura de dados mais complexas. Hoje é uma biblioteca razoável largamente utilizadas em diversas áreas da ciência de dados. Panda tira vantagem em utilizar outra biblioteca chamada numpy escrita em C e portanto, bastante rápida e eficiente ao lidar com dados em grandes volumes. Os seguintes formatos podem ser importados e exportados usando pandas:

Formato	**Tipo de dado**	**Le**	**Escreve**
texto	CSV	read_csv	to_csv
texto	JSON	read_json	to_json
texto	HTML	read_html	to_html
texto	Local clipboard	read_clipboard	to_clipboard
binário	MS Excel	read_excel	to_excel
binário	HDF5 Format	read_hdf	to_hdf
binário	Feather Format	read_feather	to_feather
binário	Msgpack	read_msgpack	to_msgpack
binário	Stata	read_stata	to_stata
binário	SAS	read_sas	
binário	Python Pickle Format	read_pickle	to_pickle
SQL	SQL	read_sql	to_sql
SQL	Google Big Query	read_gbq	to_gbq

Baixe o arquivo **assay.csv** e inicie o python.

Primeiramente vamos trabalhar com pandas lendo o conteúdo desse arquivo.

```
>>> import pandas as pd
>>> data = pd.read_csv('assay.csv')
>>> data.head()
 HOLEID FROM  TO  AU
0 M001  0.0  2.5 0.00
1 M001  2.5  5.0 0.71
2 M001  5.0  7.5 0.96
3 M001  7.5 10.0 0.48
4 M001 10.0 12.5 1.42]
>>> type(data)
<class 'pandas.core.frame.DataFrame'>
>>> len(data)
17664
>>> data.shape
(17664, 4)
>>> data.columns.values
array(['HOLEID', 'FROM', 'TO', 'AU'], dtype=object)
>>> data.dtypes
HOLEID   object
FROM    float64
TO      float64
AU      float64
dtype: object
```

Podemos selecionar colunas de dados da seguinte forma:

```
>>> selecao=data[['FROM','AU']]
```

```
>>> selecao.head()
  FROM   AU
0  0.0 0.00
1  2.5 0.71
2  5.0 0.96
3  7.5 0.48
4 10.0 1.42
```
Aplicando estatística descritiva nos dados
```
>>> selecao.mean()
pressao_min 940.133461
FROM   159.362913
AU     0.673208
dtype: float64
>>> selecao.max()
FROM   497.5
AU     30.5
dtype: float64
>>> selecao.min()
FROM   0.0
AU     0.0
dtype: float64
>>> selecao.median()
FROM   144.40
AU     0.34
dtype: float64
>>> selecao.std()
FROM   113.517974
AU     1.138792
dtype: float64
>>> selecao.describe()
                FROM             AU
count  17664.000000  17664.000000
mean     159.362913      0.673208
std      113.517974      1.138792
min        0.000000      0.000000
25%       65.000000      0.100000
50%      144.400000      0.340000
75%      227.500000      0.802500
max      497.500000     30.500000
```
Vamos agora usar pandas para fazer o caminho inverso, de lista de dados para arquivo
Um arquivo chamado estacoes.csv.
```
>>> estacao=['E-01','E-02','E-03','E-04','E-05','E-06']
>>> latitude=[-3.34,-3.23,-3.12,-3.32,-3.33,-3.19]
>>> longitude=[-60.12,-60.43,-60.11,-60.54,-59.87,-60.00]
>>> dadoEst = pd.DataFrame(data = {"Estação" : estacao, "latitude"
:latitude,"longitude" : longitude})
>>> dadoEst
Estação latitude longitude
0 E-01 -3.34 -60.12
```

```
1 E-02 -3.23 -60.43
2 E-03 -3.12 -60.11
3 E-04 -3.32 -60.54
4 E-05 -3.33 -59.87
5 E-06 -3.19 -60.00
>>> dadoEst.to_csv('estacoes.csv')
```

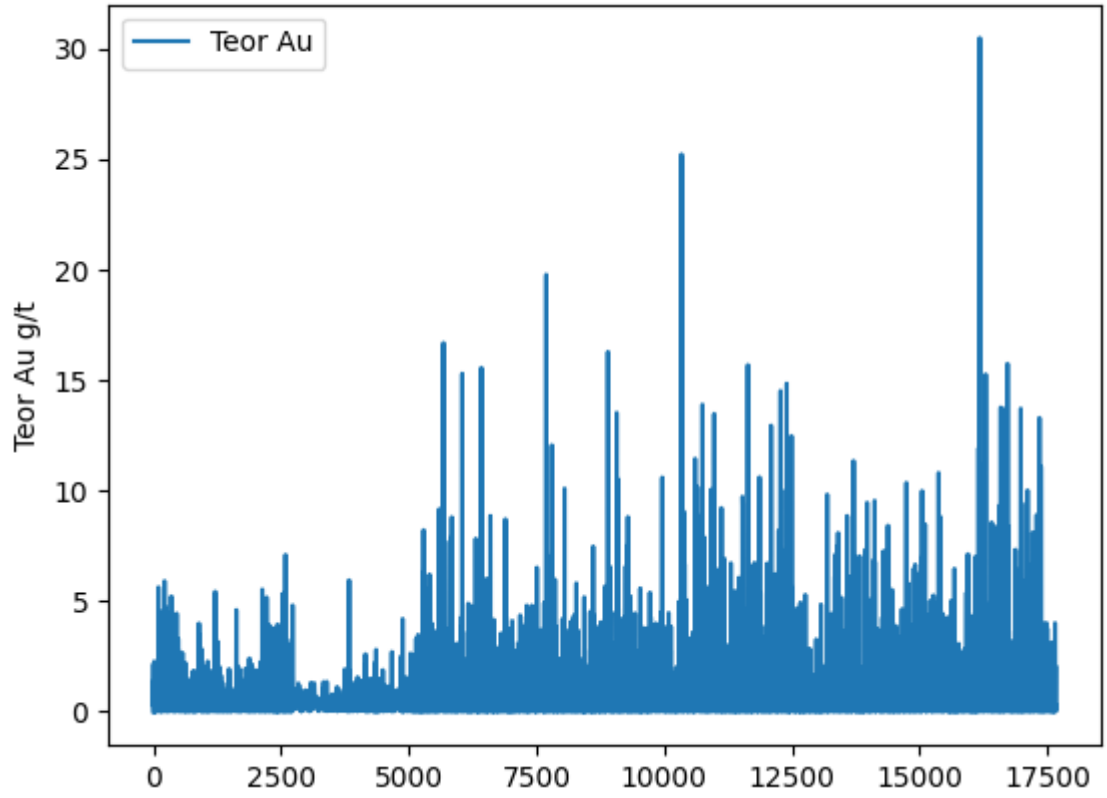
Por último, criando um data frame vazio.

```
>>> df = pd.DataFrame()
>>> print(df)
Empty DataFrame
Columns: []
Index: []
```

6 - Matplotlib

Vamos mostrar simplificadamente como podemos criar gráficos com python. Existem diversas bibliotecas para a criação de gráficos mas aqui vamos usar o matplotlib com o auxílio de pandas para criar alguns gráficos básicos.

```
>>> import pandas as pd
>>> import matplotlib.pyplot as plt
>>> data = pd.read_csv('assay.csv')
>>> plt.plot(data[['AU']], label='Teor Au')
[<matplotlib.lines.Line2D object at 0x7f55bbc70d90>]
>>> plt.ylabel('Teor Au g/t')
Text(0, 0.5, 'Precipitação mm')
>>> plt.legend()
<matplotlib.legend.Legend object at 0x7f55bcee2b50>
>>> plt.show()
```



Vamos usar o numpy para criar uma série sequencial de números entre 0 e 2 e plotar a sequência linear, quadrática e cúbica da série para ilustrar como criar um gráfico com mais de uma curva.

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> x = np.linspace(0, 2, 100)
>>> plt.plot(x, x, label='linear')
>>> plt.plot(x, x**2, label='quadrática')
>>> plt.plot(x, x**3, label='cúbica')
>>> plt.ylabel('Eixo Y')
>>> plt.xlabel('Eixo X')
>>> plt.title("Gráfico Simples")
>>> plt.legend()
```

```
>>> plt.show()
```

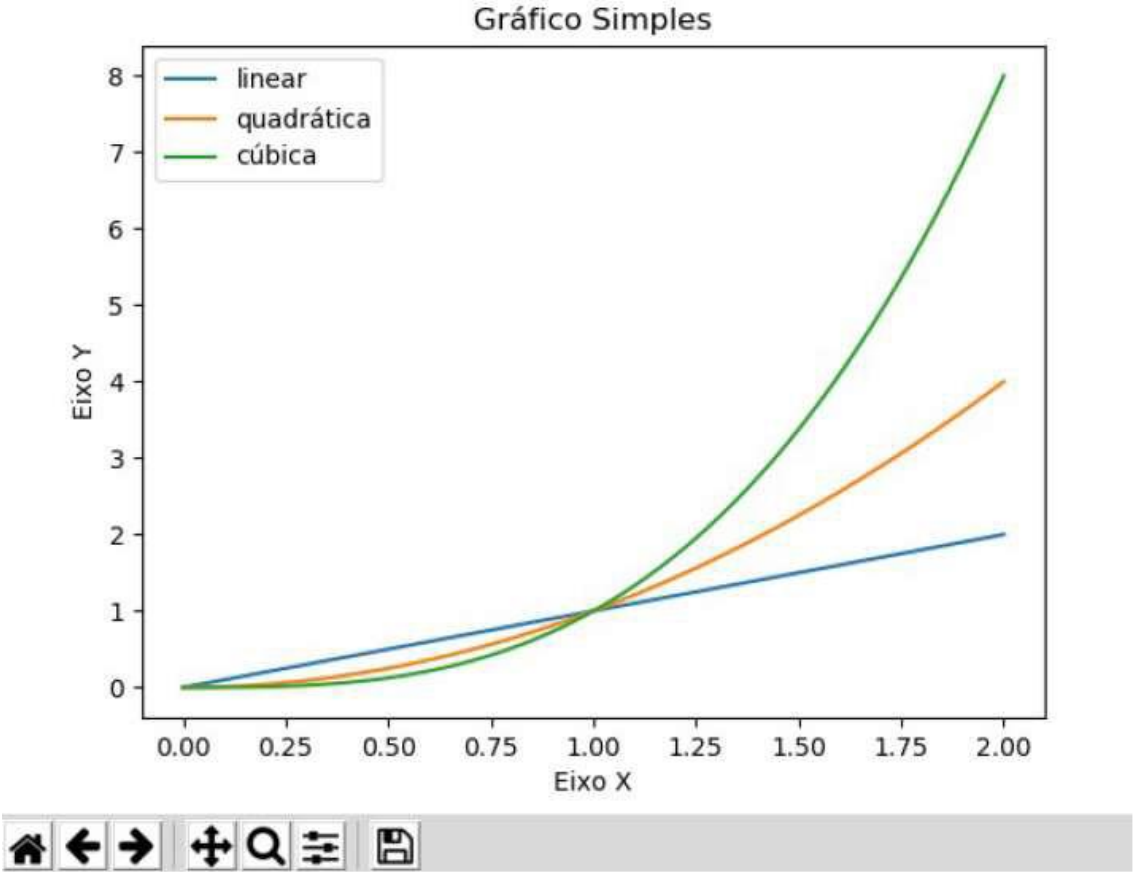


Gráfico Simples
linear
quadrática
cúbica
Eixo Y
Eixo X
0
1
2
3
4
5
6
7
8
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00