

Introduction to pyQGIS



Module 1

André Luiz Lima Costa, P.Geo., FAIG, CBRR

4th edition - October 2026

Data

This publication and the data are available at:

<https://gdatasystems.com/repo/pyQGISC1.pdf>

<https://gdatasystems.com.br/learn.php>

VISIT:

<https://gdatasystems.com>

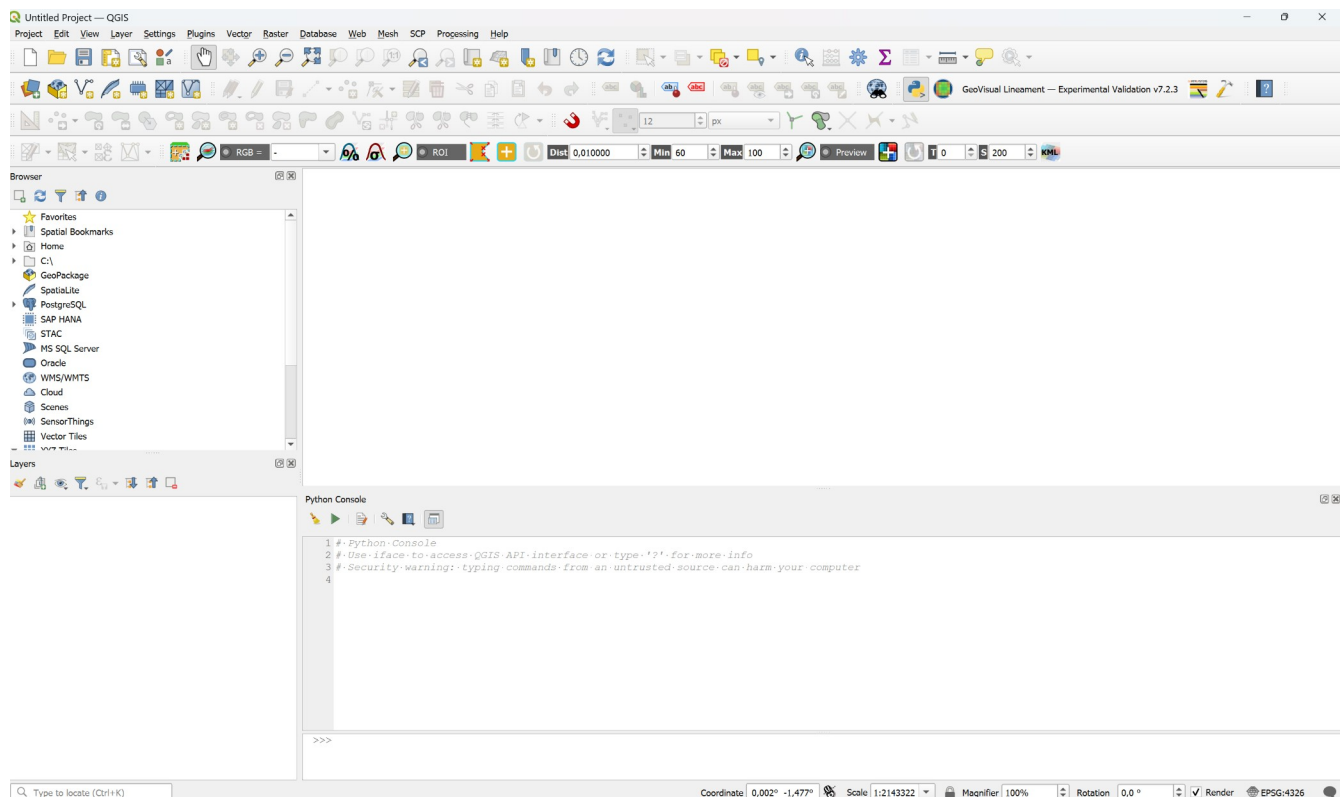
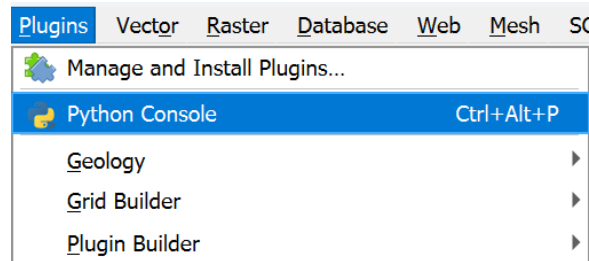
License : CC BY 4.0

<https://creativecommons.org/licenses/by/4.0/>

Module 1 - Learning Python in QGIS Using Python Console or OSGeo4W Terminal

1 – Python Console

To launch the python Console in QGIS use the menu Plugins → Python Console or type **Ctrl-Alt-P**.



Nesse tutorial de introdução à linguagem python usaremos 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.



```
OSGeo4W Shell - python
run o-help for a list of available commands
C:\Program Files\QGIS 3.34.1>python
Python 3.9.5 (tags/v3.9.5:0a7dcdb, May  3 2021, 17:27:52) [MSC v.1928 64 bit (AMD64)] on
win32
Type "help", "copyright", "credits" or "license" for more information.
>>> |
```

2 - Fundamentals of the Python language

Script comments in Python begin with #.

```
>>> # a comment
>>> print('QGIS')
QGIS
>>> a = '# This is not a comment because it is in quotation marks.'
```

Numeric Types

Python natively includes the following numeric types: integers (int), floating-point numbers (float), and complex numbers (complex).

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

The `type(variable)` function returns the type of the variable.

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

The result of a division will always be of type float.

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

The console can also function as a calculator.

```
>>> 2+2
4
>>> 2**8 #two to the power of eight
256
>>> 82%3 #integer remainder of the division
1
>>> 82//3 #quotient of the division
27
>>> 3-2
1
>>> (50-5*6)/4 # Parentheses take precedence in the operation
5.0
```

Alphanumeric Type

In Python, the `'str'` type is used for words, text, and alphanumeric characters.

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

An object of the `'str'` class is essentially an array of sequential values, where the first value (character) corresponds to index 0 of the array, the last value to index -1, the second-to-last to -2, and so on.

```
>>> nome[0]
'M'
>>> nome[1]
'a'
>>> nome[-1]
```

```
'l'
>>> nome[-2]
'e'
>>> nome[-6]
'M'note that the value at index 5 is not included
>>> nome[2:5] #note que o valor do índice 5 não está incluído
'nue'
```

Once defined, a `str` object is immutable; if we attempt to change the value of a `str` object, an error message will appear.

```
>>> 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
```

The `len()` function returns the length of the str object.

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

We can use single or double quotes to delimit a `str` object. This is convenient for defining strings that contain these characters:

```
>>> s1 = "Good morning, Mr. O'Brien"
>>> print(s1)
Good morning, Mr. O'Brien
>>> s2= 'Quoth the raven "Nevermore". '
>>> print(s2)
Quoth the raven "Nevermore"
```

List Type

Python has several compound data types; the list is one of them.

A list is a collection of objects—either of the same type or of different types—stored together.

Lists are declared within square brackets, with each object (item) separated by a comma.

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

Just like with a string (str), each item in a list can be accessed using indices.

```
>>> 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
```

Unlike the str object, the values of items in a list can be modified.

```
>>> caipi[3]='pinga'
>>> caipi
['geló', 'limão', 'açúcar', 'pinga', 15.99]
```

A list can contain other lists as items. And we access each item within that inner list

using an additional index.

```
>>> caipi[1]=['abacaxi','maracujá','limão']
>>> caipi
['gelo', ['abacaxi', 'maracujá', 'limão'], 'açúcar', 'pinga', 15.99]
>>> caipi[1][1]
'maracujá'
>>> caipi[1][-1]
'limão'
```

We can add items to an existing list using addition or the append() function.

```
>>> caipi = caipi + ['guardanapo','canudo']
>>> caipi.append('copo')
>>> caipi
['gelo', ['abacaxi', 'maracujá', 'limão'], 'açúcar', 'pinga', 15.99, 'guardanapo', 'canudo', 'copo']
```

Some methods for interacting with lists are shown below:

We begin by creating the following empty list.

```
>>> lis=[]
>>> lis
[]
```

Adding items to the list using the extend() method.

```
>>> lis.extend([1,2,3])
>>> lis
[1, 2, 3]
```

Adding an item with a value of 10 at the predetermined position (0) using the insert() method.

```
>>> lis.insert(0,10)
>>> lis
[10, 1, 2, 3]
```

Removing the first occurrence of the value passed to the remove() method from the list.

```
>>> lis.remove(2)
>>> lis
[10, 1, 3]
```

We can copy a list using the copy() method.

```
>>> lis2=lis.copy()
>>> lis2
[10, 1, 3]
```

We reverse the order of the items in a list using the reverse() method.

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

We sort a list using the sort() method.

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

The pop() method removes and returns the item at the specified index; if no index is provided, the last item is removed from the list and returned.

```
>>> 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]
```

The index(x[,start[,end]]) method returns the (zero-based) index of the first occurrence of item x. The second and third arguments specify the range—from which index to start and at which index to stop—within the list. If an item with the specified value is not found, an error message is returned.

```
>>> 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

```

The count(x) method returns the number of times the item x appears in the list.

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

```
3
```

The clear() method removes all items from the list.

```
>>> lis3.clear()
```

```
>>> lis3
```

```
[]
```

We can use the `del` statement to delete items from a list using indices instead of values.

```
>>> 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

Much like strings and lists, tuples are sequence data types used in Python. The main differences between a tuple and a list are that tuples are defined using commas to separate their items, and these items are immutable.

```
>>> 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 are another type of sequential data structure used in Python. They are defined within curly braces {} and ensure that each item appears only once; furthermore, they are not indexed.

```
>>> 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'}
```

Dictionaries

Dictionaries are a data type frequently used in Python. A dictionary always consists of a key and a value; the key serves the same purpose as the numeric index used in a list. Each key must be unique, and an empty dictionary can be created using {}.

```
>>> 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}
```

Internal Functions

The following built-in functions are used for type conversion and type information.

```
>>> 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

```

The following words are reserved in the Python language and cannot be used to define variables.

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	

And these are the internal functions:

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()	

3 - Flow controls

Like any programming language, Python uses program flow controls. But before we get to that, let's talk a bit about script indentation. In Python, the use of indentation for code blocks is mandatory, as we do not separate them using curly braces {}. Therefore, every code block—whether it is a class, a function, or a flow control structure—must be indented. When working with an indented block in the console, the `>>>` prompt changes to `...`, indicating that you are inside a block within the script. Once all the statements in the block are complete, you press 'Enter' on the final line (the one showing `...`).

if elif else

The `if` statement is perhaps the best-known instruction in programming. It checks whether a condition—or, alternatively, other conditions (`elif`)—are met. If no conditions are met, we can also specify an action to be taken (`else`).

```
>>> x = int(input("Name an integer: "))
```

Diga um número inteiro: 2

```
>>> if x<0:
...     print('The number is negative')
... elif x>0:
...     print('The number is positive')
... else:
...     print('The number is zero')
...
The number is positive
```

for

The `for` loop is defined as “executing/repeating instructions according to predefined terms.” In the example below, the loop iterates over each word in the variable `palavras`, and both the word and its length are printed as the result.

```
>>> palavras = ['Hi!', 'Lets', 'learn', 'python?']
>>> for palavra in palavras:
...     print(palavra, len(palavra))
...
Hi! 4
Lets 5
learn 8
python? 7
```

while

The `while` loop is defined as: “as long as a predefined condition has not been met, keep executing/repeating.”

```
>>> 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()

The way Python interacts with a numerical sequence is by using the range() function.

```
>>> for i in range(10):
...     print(i, end=', ')
...
0,1,2,3,4,5,6,7,8,9,
```

Let's assume we don't know the number of items in a list, but we want to iterate through (in this case, list) each of these items. We use `range` to help us.

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

We can define the start and end of range(), as well as the step.

```
>>> 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 and else in loops

The `break` statement terminates the execution of the innermost `for` or `while` loop.

Loops can also have an `else` clause; it is executed when the loop finishes, but not when it is interrupted by a `break` statement. See the example below:

```
>>> for n in range(2, 10):
...     for x in range(2, n):
...         if n % x == 0:
...             print(n, 'it is equal to ', x, '*', n//x)
...             break
...         else:
...             print(n, 'it is a prime number')
...
2 it is a prime number
3 it is a prime number
4 t is equal to 2 * 2
5 it is a prime number
6 t is equal to 2 * 3
7 it is a prime number
8 t is equal to 2 * 4
9 t is equal to 3 * 3
```

The `continue` statement advances to the next iteration of the loop:

```
>>> for num in range(2, 6):
...     if num % 2 == 0:
...         print("Even Number", num)
...         continue
...     print("Odd Number", num)
...
Even Number 2
Odd Number 3
Even Number 4
Odd Number 5
```

4 - Functions

A function is a set of organized, reusable instructions for performing a task. In Python, we define a function using `def`, followed by the function name and any arguments or parameters enclosed in parentheses. See the example below:

```
>>> def fibo(n):
...     """Calculates the Fibonacci sequence up to the specified limit"""
```

```

... 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

```

Another way to write this function, returning the result in list format, is shown below:

```

>>> def fibo2(n):
...     """Calculates and returns the Fibonacci sequence up to the specified limit."""
...     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]

```

We can assign a function to a variable.

```

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

```

If we pass our function as an argument to the `help` function, we get the documentation string (enclosed within triple double quotes) as the returned information. This is one way to document what a function does in Python.

```

>>> help(fibo2)
Help on function fibo2 in module __main__:
fibo2(n)
Calculates and returns the Fibonacci sequence up to the specified limit.

```

It is possible to define functions with a variable number of arguments.

Using predefined default values is the most common approach, where we can set a default value for one or more of a function's parameters.

```

>>> def pergunta(texto, tentativas=4, msg='Try again!'):
...     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('Invalid Answer.')
...         print(msg)
...
>>> pergunta('Are you hungry?')
Are you hungry?j
Try again!
Are you hungry?Sim
True
>>> pergunta('Are you hungry?',1,'I didn't understand the answer!')
Está com fome?iop
I didn't understand the answer!

```

```
Are you hungry?poi
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
File "<stdin>", line 10, in pergunta
ValueError: Invalid Answer.
```

Now, let's look at a practical example of how to work with functions by creating our first script. Create a file named `temp_converter.py` with the following code:

```
#!/usr/bin/env python3
'''Converts temperatures from Celsius to Fahrenheit, Kelvin to
Celsius, and Kelvin
to Fahrenheit.
Usage:
Load using import
import temp_converter as tc
Author:
Your Name - 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
```

Now let's import our script and use functions from it.

```
>>> import temp_converter as tc
>>> print("The freezing point of water in Fahrenheit is:",
tc.celsius_para_fahrenheit(0))
The freezing point of water in Fahrenheit is: 32.0
>>> print('Absolute zero in Fahrenheit is:',
tc.kelvins_para_fahr(0))
Absolute zero in Fahrenheit is: -459.66999999999996
```

5 - Modules

In Python, a module is simply a file with the `.py` extension containing classes, functions, and other instructions. Our script above is an example of a module. A package is a way to organize several modules into a larger entity. Some languages call modules and packages libraries. A module is loaded using the `import` command, and we can rename a module using `'as'`

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

We can also import a simple function from a module using `'from'`.

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

Or we can import a submodule from a module.

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

We will be using various modules throughout; this is the strength of the Python language, which offers countless existing modules for a wide range of functions. We will begin by looking at the `pandas` and `matplotlib` modules..

6 - Pandas

The pandas library was developed as an alternative to the R language for handling more complex data structures. Today, it is a widely used library across various fields of data science. Pandas leverages another library called NumPy, which is written in C and is therefore highly fast and efficient when dealing with large volumes of data. The following formats can be imported and exported using pandas:

Format	Data Type	Read	Write
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

Download the file assay.csv and start Python.

First, we will work with pandas by reading the contents of this file.

```
>>> 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
```

We can select data columns as follows:

```
>>> 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
```

Applying descriptive statistics to the data:

```
>>> 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
```

Now, let's use pandas to perform the reverse process: going from a list of data to a file—specifically, a file named `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
estacao  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')
```

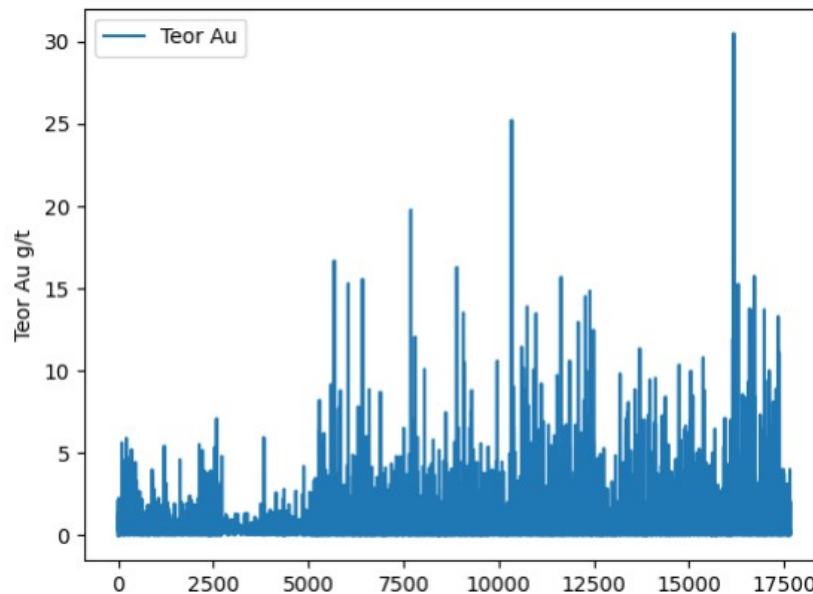
Lastly, creating an empty data frame.

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

7 - Matplotlib

We will show, in a simplified way, how to create charts using Python. There are several libraries available for creating charts, but here we will use Matplotlib, with the help of Pandas, to create some basic charts.

```
>>> 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()
```



We will use NumPy to create a sequential series of numbers between 0 and 2, and then plot the linear, quadratic, and cubic sequences derived from this series to illustrate how to create a graph containing more than one curve.

```
>>> 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='quadratic')
>>> plt.plot(x, x**3, label='cubic')
>>> plt.ylabel('Eixo Y')
>>> plt.xlabel('Eixo X')
>>> plt.title("Simple Chart")
>>> plt.legend()
>>> plt.show()
```

