Introduction to Python

Ruihan Lu

GradQuant
University of California, Riverside, CA

February 10, 2021

Table of Contents

Table of Contents

Background

Python is a popular programming language. It was created by Guido van Rossum, and released in 1991.
It is used for:

- web development (server-side);
- software development;
- mathematics;
- system scripting.

What Python Can do

- Python can be used on a server to create web applications;
- Python can be used alongside software to create workflows;
- Python can connect to database systems. It can also read and modify files;
- Python can be used to handle big data and perform complex mathematics;
- Python can be used for rapid prototyping, or for production-ready software development

Why Python

- Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc);
- Python has a simple syntax similar to the English language;
- Python has syntax that allows developers to write programs with fewer lines than some other programming languages;
- Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick;
- Python can be treated in a procedural way, an object-oriented way or a functional way

Table of Contents

Comment

Comments can be used to explain Python code and make the code more readable.

```
print('Hello World!') #comment
```

```
Hello World!
```

Create Variables

- Variables are containers for storing data values;
- Variables do not need to be declared with any particular type, and can even change type after they have been set;
- A variable name must start with a letter or the underscore character;
- A variable name **cannot** start with a number;
- A variable name can only contain alpha-numeric characters and underscores;
- Variable names are case-sensitive.

Create Variables

```
x = 5
print(x)
```

```
5
```

-

```
y = 'Hello'
print(y)
```

```
Hello
```

-

```
a = 4
A = "Sally"
```

```
print(a)
print(A)
```

```
4
```

Create Variables

Python allows you to assign values to multiple variables in one line, **but** make sure the number of variables matches the number of values, or else you will get an error.

```
x, y, z = "Orange", "Banana", "Cherry"
print(x)
print(y)
print(z)
```

```
Orange
Banana
Cherry
```

Create Variables

To combine both text and a variable, Python uses the + character.

```
x = "awesome"
print("Python is " + x)
```

```
Python is awesome
```

Create Variables

You can also use the + character to add a variable to another variable.

```
x = "Python is "
y = "awesome"
z =  x + y
print(z)
```

```
Python is awesome
```

Create Variables

For numbers, the + character works as a mathematical operator.

```
x = 5
y = 10
print(x + y)
```

- 15

Data Types

Variables can store data of different types, and different types can do different things.
Python has the following data types built-in by default, in these categories:

- **Text Type**: str ;
- **Numeric Types**: int, float, complex ;
- **Sequence Types**: list, tuple, range;
- **Mapping Type**: dict;
- **Boolean Type**: bool;
- **Binary Types**: bytes, bytearray, memoryview

Data Types

You can get the data type of any object by using the **type()** function.

```
x1 = "Hello World"
x2 = 20
x3 = 20.5
x4 = 1j
print(type(x1))
print(type(x2))
print(type(x3))
print(type(x4))
```

```
<class 'str'>
<class 'int'>
<class 'float'>
<class 'complex'>
```

Data Types

```
x5 = ["apple", "banana", "cherry"]
x6 = ("apple", "banana", "cherry")
x7 = range(6)
x8 = {"name" : "John", "age" : 36}
x9 = True
print(type(x5))
print(type(x6))
print(type(x7))
print(type(x8))
print(type(x9))
```

```
<class 'list'>
<class 'tuple'>
<class 'range'>
<class 'dict'>
<class 'bool'>
```

Data Types

We can convert from one data type to another.

```
x = 1    # int
y = 2.8  # float

#convert from int to float:
a = float(x)

#convert from float to complex:
b = complex(y)

print(a)
print(b)

print(type(a))
print(type(b))
```

```
1.0
(2.8+0j)
<class 'float'>
<class 'complex'>
```

Table of Contents

Built-in Math Functions

Python has a set of built-in math functions, including an extensive math module, that allows you to perform mathematical tasks on numbers.

- The **min()** and **max()** functions can be used to find the lowest or highest value in a vector.

```
x = min(5, 10, 25)
y = max(5, 10, 25)

print(x)
print(y)
```

```
5
25
```

Built-in Math Functions

- The **abs()** function returns the absolute (positive) value of the specified number.

```
x = abs(-7.25)

print(x)
```

```
7.25
```

- The **pow(x, y)** function returns the value of x to the power of y, which is x^y.

```
x = pow(4, 3)

print(x)
```

```
64
```

Built-in Math Module

Python has also a built-in module called **math**, which extends the list of mathematical functions. To use it, you must import the **math** module. The **sqrt()** function in **math** module returns the square root of a number.

```
import math
x = math.sqrt(64)

print(x)
```

```
8.0
```

Built-in Math Module

The **ceil()** method rounds a number upwards to its nearest integer, and the **floor()** method rounds a number downwards to its nearest integer.

```
x = math.ceil(1.4)
y = math.floor(1.4)

print(x)
print(y)
```

```
2
1
```

Table of Contents

Conditions

Python supports the usual logical conditions from mathematics:

- **Equals**: a == b ;
- **Not Equals**: a != b;
- **Less than**: a $<$ b ;
- **Less than or equal to**: a $\leq$ b ;
- **Greater than**: a $>$ b;
- **Greater than or equal to**: a $\geq$ b.

IF Statement

An 'IF statement' is written by using the **if** keyword.

```
a = 33
b = 200
if b > a:
    print("b is greater than a")
```

- `b is greater than a`

The **elif** keyword is pythons way of saying 'if the previous conditions were not true, then try this condition'.

```
a = 33
b = 33
if b > a:
    print("b is greater than a")
elif a == b:
    print("a and b are equal")
```

- `a and b are equal`

IF Statement

The **else** keyword catches anything which isn't caught by the preceding conditions.

```
a = 200
b = 33
if b > a:
    print("b is greater than a")
elif a == b:
    print("a and b are equal")
else:
    print("a is greater than b")
```

- a is greater than b

Logical Operators

The **and** and **or** keyword are logical operators, and are used to combine conditional statements.

```
a = 200
b = 33
c = 500
if a > b and c > a:
    print("Both conditions are True")
```

- ```
  Both conditions are True
  ```

```
a = 200
b = 33
c = 500
if a > b or a > c:
    print("At least one of the conditions is True")
```

- ```
  At least one of the conditions is True
  ```

Nested IF

You can have **if** statements inside **if** statements, this is called **nested if** statements.

```
x = 41

if x > 10:
    print("Above ten,")
    if x > 20:
        print("and also above 20!")
    else:
        print("but not above 20.")
```

```
Above ten,
and also above 20!
```

Pass Statement

If statements cannot be empty, but if you for some reason have an **if** statement with no content, put in the **pass** statement to avoid getting an error.

```
a = 33
b = 200

if b > a:
    pass
```


For Loop

A **for** loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string)

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
    print(x)
```

```
apple
banana
cherry
```

For Loop

With the **break** statement we can stop the loop before it has looped through all the items.

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
    print(x)
    if x == "banana":
        break
```

```
apple
banana
```

For Loop

- Loop through a set of specified number of times, we can use the **range()** function;
- The **range()** function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number;
- Note that range(6) is not the values of 0 to 6, but the values 0 to 5.

```
for x in range(2, 6):
    print(x)
```

```
2
3
4
5
```

Create a Function

In Python a function is defined using the **def** keyword.

- Information can be passed into functions as arguments;
- Arguments are specified after the function name, inside the parentheses.

```
def my_function(fname):
    print(fname + " UCR")

my_function("Emily")
my_function("Lily")
```

```
Emily UCR
Lily UCR
```

Create a Function

- You can add as many arguments as you want, just separate them with a comma.

```
def my_function2(x, y):
    print(x+y)
```

```
my_function2(3, 5)
```

```
8
```

```
def my_function3(x, y, z):
    print((x+y)*z)

my_function3(1, 2, 3)
```

```
9
```

Table of Contents

Import Data Files

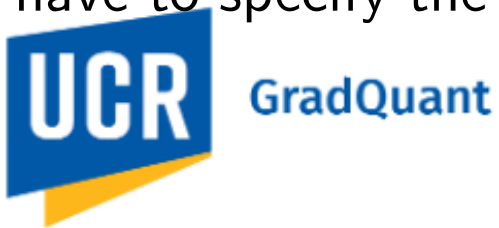
Python has built-in functions to read in data files.

- Assume we have the following file, located in the same folder as Python.
- To open the file, use the built-in **open()** function.
- The **open()** function returns a file object, which has a **read()** method for reading the content of the file.

```
f = open("sample.txt", "r")
print(f.read())
```

```
Sample text for Python workshop
```

- If the file is located in a different location, you will have to specify the file path.



Import Data Files

- By default the read() method returns the whole text, but you can also specify how many characters you want to return.

```
f = open("sample.txt", "r")
print(f.read(5))
```

```
Sampl
```

Manipulate Files

You also can manipulate an existing file in Python. to write an existing file, you must add a parameter to the **open()** function:

- **'a'**, which refers to append. Will append to the end of the file.

```
f = open("sample.txt", "a")
f.write("Now the file has more content!")
f.close()

#open and read the file after the appending:
f = open("sample.txt", "r")
print(f.read())
```

```
Sample text for Python workshop
Now the file has more content!
```

Manipulate Files

- **'w'**, which refers to write. Will overwrite any existing content.

```
f = open("sample.txt", "w")
f.write("Woops! I have deleted the content!")
f.close()

#open and read the file after the appending:
f = open("sample.txt", "r")
print(f.read())
```

```
Woops! I have deleted the content!
```

Create New Files

To create a new file in Python, use the **open()** method, with one of the following parameters:

- **'x'**, which refers to create. Will create a file, returns an error if the file exist;
- **'a'**, which refers to append. Will create a file if the specified file does not exist;
- **'w'**, which refers to write. Will create a file if the specified file does not exist.

```
f = open("myfile.txt", "x")
```

Table of Contents

Useful Library

- **NumPy** is a Python library used for working with arrays;
- **NumPy** stands for Numerical Python;
- It also has functions for working in domain of linear algebra, Fourier transform, and matrices;
- **NumPy** was created in 2005 and it is an open source project and you can use it freely.
- **NumPy** aims to provide an array object that is up to 50x faster than traditional Python lists.

Useful Library

- **Pandas** is a Python library used for working with data sets;
- It has functions for analyzing, cleaning, exploring, and manipulating data;
- **Pandas** allows us to analyze big data and make conclusions based on statistical theories;
- **Pandas** can clean messy data sets, and make them readable and relevant

Useful Library

- **SciPy** is a scientific computation library that uses **NumPy** underneath. It offers additional functionality compared to **NumPy**, including **scipy.stats** for statistical analysis;
- **SciPy** stands for Scientific Python. It provides more utility functions for optimization, stats and signal processing;
- **SciPy** is open source so we can use it freely.

Examples for Statistical Analysis

Pandas can input data from SQL, excel files, or other formats.

```
import pandas as pd
```

```
data = pd.read_csv('test.csv', sep=',', na_values=".")
data
```

Examples

describe function allows you to do basic statistical summary for each column in your dataset.

```
data.describe()
```

	ID	Label	Y	Z1	Z2	X1	X2
count	11.0	11.000000	11.000000	11.000000	11.000000	11.000000	11.000000
mean	1.0	1.545455	1.457856	0.454545	-0.717345	0.640513	-1.864769
std	0.0	0.522233	10.290188	0.522233	0.680010	4.358998	4.163404
min	1.0	1.000000	-15.607313	0.000000	-1.324485	-5.575913	-8.194061
25%	1.0	1.000000	-3.410064	0.000000	-1.324485	-1.975133	-4.593331
50%	1.0	2.000000	0.130074	0.000000	-1.230026	0.480074	-1.279101
75%	1.0	2.000000	3.922257	1.000000	-0.007669	2.307325	0.471866
max	1.0	2.000000	24.136473	1.000000	-0.007669	9.872299	5.008096

dQuant

Examples

shape function returns the dimension of your dataset with first argument refers to rows and second argument refers to columns.

```
data.shape
```

```
(11, 7)
```

columns function returns all the column names in your dataset.

```
data.columns
```

```
Index(['ID', 'Label', 'Y', 'Z1', 'Z2', 'X1', 'X2'], dtype='object')
```

Examples

If you have any categorical data, you can use **groupby** function to split a data frame on values of categorical variables.

```
groupby_label = data.groupby('Label')
groupby_label.mean()
```

	ID	Y	Z1	Z2	X1	X2
Label						
1	1	3.576623	1	-0.007669	1.499954	-2.556275
2	1	-0.307783	0	-1.308742	-0.075688	-1.288515

-

```
for Label, value in groupby_label['Y']:
    print((Label, value.mean()))
```

```
(1, 3.5766232654)
(2, -0.3077830079999964)
```

-

Examples

For simple statistical tests, we will use the **scipy.stats** sub-module of **scipy**.

- For a One-sample T test use **scipy.stats.ttest1samp()** function. It tests if the population mean of data is likely to be equal to a given value (technically if observations are drawn from a Gaussian distributions of given population mean). It returns the T statistic, and the p-value

```
stats.ttest_1samp(data['Y'], 0)
```

```
Ttest_1sampResult(statistic=0.46988083976065675, pvalue=0.6485193148827186)
```

- With a p-value equals to 0.65 that we can claim that the population mean for our Y is not significantly different from 0.

Examples

- For a Two-sample T test use **scipy.stats.ttestind()** function. It tests if there exists any difference for the population mean across two groups.

```
label1 = data[data['Label'] == 1]['X1']
label2 = data[data['Label'] == 2]['X1']
stats.ttest_ind(label1, label2)
```

```
Ttest_indResult(statistic=0.5766809564427937, pvalue=0.5782910379829181)
```

  - With the p-value equals to 0.58, we can claim that there does not exists any significant difference for the population mean of our X_1 variables among label 1 and label 2.

Examples

We will use the **statsmodels** module to do linear regression. It can fit a linear model as well as test that whether coefficient is significant or not.

```
from statsmodels.formula.api import ols
```

```
lineardata = pd.read_csv('Salary_Data.csv')
```

```
lineardata.columns
```

```
Index(['YearsExperience', 'Salary'], dtype='object')
```

Examples

Let's fit a simple linear model to our data and test whether the coefficient are significant or not.

```
model = ols("Salary ~ YearsExperience", lineardata).fit()
```

```
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                 Salary   R-squared:                       0.957
Model:                            OLS   Adj. R-squared:                  0.955
Method:                 Least Squares   F-statistic:                     622.5
Date:                Sun, 07 Feb 2021   Prob (F-statistic):           1.14e-20
Time:                        23:21:42   Log-Likelihood:                -301.44
No. Observations:                  30   AIC:                             606.9
Df Residuals:                      28   BIC:                             609.7
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      2.579e+04   2273.053     11.347      0.000    2.11e+04    3.04e+04
YearsExperience  9449.9623    378.755     24.950      0.000    8674.119    1.02e+04
```

-

Services in GradQuant

- GradQuant offers individual consultations on Skype. Always welcome to make an appointment with us.
- Weekly Drop-in Hours on Tuesday from 9 am to 11 am. Graduate students and postdocs to meet with GradQuant consultants without an appointment, on a 'first-come, first-served' basis.
- Weekly Hacky Hours on Thursday from 11 am to 1 pm. Hacky Hours open to the whole campus, serving undergraduate and graduate students, faculty, and staff. No appointment is required.
- For detailed information about how to make an appointment and attend drop-in hours, you can visit via `https://gradquant.ucr.edu`.

Question?