Python3 Quick Start Guide

Saleh Zare Zade salehz@wayne.edu

1. Install Python

1.1 Install Python in Windows

To install Python, you must first download the installation package of your preferred version from this link: https://www.python.org/downloads/. On this page, you will be asked to choose between different versions, the latest versions for Python 3 is Python 3.7.4. Alternatively, if you are looking for a specific release, you can scroll down the page to find download links for earlier versions.

1.2 Installing Python in Mac

If you're using a Mac, you can download the installation package from this link: https://www.python.org/downloads/mac-osx/.

1.3 Installing Python In IDE

I highly recommend using some popular Python IDE.
Anaconda https://www.anaconda.com/
PyCharm http://www.jetbrains.com/pycharm/download/#section=windows
Spyder https://docs.spyder-ide.org/
Jupyter Notebook https://jupyter.org/

2. Variables

A variable is like a container that stores values that you can access or change. Python differs significantly from languages such as Java, C, or C++ when it comes to dealing with variables. Other languages declare and bind a variable to a specific data type. This means that it can only store a unique data type. Hence, if a variable is of integer type, you can only save integers in that variable when running your program. Python is a lot more flexible when it comes to handling variables. If you need a variable, you'll just think of a name and declare it by assigning a value. If you need to, you can change the value and data type that the variable stores during program execution. For example:

```
my_variable = 10
print(my_variable)
my_variable = my_variable + 3
print(my_variable)
my_variable = "yellow"
print(my_variable)
```

```
10
13
yellow
```

3. Data Types

3.1 String

A string is a sequence of Unicode characters that may be a combination of letters, numbers, and special symbols. To define a string in Python, you can enclose the string in matching single or double quotes:

```
s = "Hello Python"
print(s)
print(len(s))  # get the length of s
print(s[0])    # get the first element in s
print(s[2:6])  # slice s
print(s.lower()) # get the lower format of s
print(s.upper()) # get the upper format of s
```

```
Hello Python
12
H
llo
hello python
HELLO PYTHON
```

3.2 Tuple

A tuple is a sequence of elements in order, it is expressed in parentheses, or without parentheses.

```
a_tuple = (12, 3, 5, 15 , 6)
another_tuple = 12, 3, 5, 15 , 6
```

3.3 List

A list is also a sequence of elements in order, but it is expressed in bracket.

```
a_list = [12, 3, 67, 7, 82]
```

Some functions of list:

append

```
a = [1,2,3,4]
a.append(0)  # Append a with 0 in the end
print(a)
```

```
[1, 2, 3, 4, 0]
```

insert

```
a = [1,2,3,4]
a.insert(1,0) # Add a 0 at position 1
print(a)
```

```
[1, 0, 2, 3, 4]
```

remove

```
a = [1,2,3,4,2]
a.remove(2) # Delete the first item in the list that appears with a value of 2
print(a)
```

```
[1, 3, 4, 2]
```

index

```
a = [1, 2, 3, 4]
print(a[0])  # Displays the 0th elment of list a
print(a[-1]) # Displays the last elmentof list a
print(a[0:3]) # Displays the element in list a from 0th through bits 2nd (before 3rd)
print(a[2:])  # Displays the 3rd element of list a and all subsequent elements
```

```
1
4
[1, 2, 3]
[3, 4]
```

```
a = [1, 2, 3, 4, 2]
print(a.index(2)) # Displays the index of the element of value 2 that first appears in list a
```

```
1
```

count

```
a = [1, 2, 3, 4, 1, 1]
print(a.count(1))  # Count the number of occurrences of a value in the list
```

```
3
```

sort

```
a = [4, 9, 3, 7, 1, 2, 5]
a.sort() # By default, sort from smallest to largest
print(a)
a.sort(reverse=True) # Sort from largest to smallest
print(a)
```

```
[1, 2, 3, 4, 5, 7, 9]
[9, 7, 5, 4, 3, 2, 1]
```

3.4 Multidimensional List (Array)

Python does not have built-in support for Arrays, but multidimensional Python lists can be used instead.

```
a = [1, 2, 3, 4, 5] # one row  and five column

multi_dim_a = [[1, 2, 3],
               [2, 3, 4],
               [3, 4, 5]] # three rows  and three columns
```

index in array

```
print(a[1])
print(multi_dim_a[0][1])
```

```
2
2
```

3.5 Dictionary

A list is sequentially output and input, while the archived form of dictionary is not sequentially. In the dictionary, there are two elements, key and value, each key corresponds to a value, key is the name, value is the content. Both numbers and strings can be used as keys or values, and all keys or values do not need to have the same form in the same dictionary.

```
dic = {'apple':1, 'pear':2, 'orange':3}
print(dic)
dic['pear'] = 20
print(dic)
```

```
{'apple': 1, 'pear': 2, 'orange': 3}
{'apple': 1, 'pear': 20, 'orange': 3}
```

4. While Loop

4.1 Basic Use

While statements are used in much the same way as in other programming languages. The main structure is as follows:

```
while condition:
    expressions
```

In this structure, condition is a judgement, which is True or False in Python. If it is True, the expressions statement will be executed, otherwise the while statement block will be skipped.

4.2 Example

For example, we want to print the numbers from 0 to 9.

```
condition = 0
while condition < 10:
    print(condition)
    condition = condition + 1
```

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

In this example, the condition is initialed as 0. In judging the condition 0 < 10, it is True, so execute the internal code in while loop. At first, print the value, then add 1 to condition, at this time, the codes finish one loop. Go back to the judgement of while loop, it is still True, repeat the previous process until the condition is 10. The judgment becomes False, it will not execute the internal code in while loop, so the number 10 could not be printed.

4.3 Attention

When we use the while loop, be careful to modify the value of the judgment condition inside the loop, otherwise the while loop will be executed forever.

```
In [*]: condition = 0
        while condition < 10:
            print(condition)
```

```
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
```

In this example, the condition is always True, so the program will print 0 all the time, in this way, we need to manually stop it!

4.4 Advanced Use

In Python, except for some normal comparison operations would return True or False, such as smaller (<), larger (>), not larger (<=), not smaller (>=), equal (==) and not equal (!=) , some other values could also return True or False.

4.4.1 Numbers

If the number is 0 or 0.0, it returns False, otherwise, it returns True.

```
condiiton = 10
while condiiton:
    print(condiiton)
    condiiton -= 1
```

```
10
9
8
7
6
5
4
3
2
1
```

4.4.2 None

If the judgment after while is type None, it will return False.

5. For Loop

5.1 Basic Use

Almost every program language has the for loop function. The basic use of for loop in Python is as follows:

```
for item in sequence:
    expressions
```

5.2 Examples

```
example_list = [1,2,3,4,5,6,7]
for i in example_list:
    print(i)
```

```
1
2
3
4
5
6
7
```

The outputs are every element in the example_list. Pay attention to the indentation in the structure of Python code. In the following example, the statement 'out of for' only print once, because it is out of the for loop structure.

```
example_list = [1,2,3]
for i in example_list:
    print(i)
    print('inner of for')
print('outer of for')
```

```
1
inner of for
2
inner of for
3
inner of for
outer of for
```

5.3 Advanced Use

5.3.1 Range

In Python, range function returns a sequence, and there are three different methods.

range (start, stop):

Start here is the beginning value of the sequence, stop is the end value, but the end value is not included in the sequence.

```
for i in range(1, 5):
    print(i)
```

```
1
2
3
4
```

range (stop):

If we omit the start, the sequence will start from 0, which is equivalent to range (0, stop).

```
for i in range(5):
    print(i)
```

```
0
1
2
3
4
```

range (start, stop, step):

Step here represents the step size, which is the difference between two values, start with start and increase the value of step until it is equal to or greater than stop.

```
for i in range(0, 10, 2):
    print(i)
```

```
0
2
4
6
8
```

5.3.2 Collections

Python has four different kinds of collections, list, tuple, dict, and set. Each collection object can iterate.

tuple:

```
tup = ('python', 3.7, 64)
for i in tup:
    print(i)
```

```
python
3.7
64
```

dict:

```
dic = {}
dic['lan'] = 'python'
dic['version'] = 3.7
dic['platform'] = 64
for key in dic:
    print(key, dic[key])
```

```
lan python
version 3.7
platform 64
```

set:

```
s = set(['python', 'python2', 'python3'])
for item in s:
    print(item)
```

```
python3
python
python2
```

6. If Statement

6.1 If Statement

6.1.1 Basic Use

Like the if statement in other programming languages, we use it in the following way:

```
if condition:
    expressions
```

If the value of condition is True, the expressions will be executed, otherwise the statement will be skipped.

6.1.2 Example

```
x = 1
y = 2
z = 3
if x < y:
    print('x is smaller than y')
```

```
x is smaller than y
```

In this example, the condition of if statement is x < y, it is True, so the print statement can be executed.

6.2 If Else Statement

6.2.1 Basic Use

```
if condition:
    true_expressions
else:
    false_expressions
```

If the condition is True, execute the true_expressions statement, while if it is False, execute the false_expressions statement.

6.2.2 Example

In the first example, the condition x>y is False, so it executes the statement of else branch. While in the second example, the condition x>y is True, the first print statement can be executed.

```
x = 1
y = 2
z = 3
if x > y:
    print('x is greater than y')
else:
    print('x is less or equal to y')
```

```
x is less or equal to y
```

```
x = 4
y = 2
z = 3
if x > y:
    print('x is greater than y')
else:
    print('x is less or equal y')
```

```
x is greater than y
```

6.3 If Elif Else Statement

6.3.1 Basic Use

```
if condition1:
    true1_expressions
elif condition2:
    true2_expressions
elif condtion3:
    true3_expressions
elif ...
    ...
else:
    else_expressions
```

If there are more than one condition, we can add more conditions through an elif statement. Once a condition is True, the corresponding expression is executed. Then jump out of the if-elif-else statement block.

6.3.2 Example

```
x = 4
if x > 1:
    print ('x > 1')
elif x < 1:
    print('x < 1')
else:
    print('x = 1')
print('finish')
```

```
x > 1
finish
```

In this example, x =4, so the first condition is True, the program will print x>1 and jump out the if-elif-else statement block, then print finish.

7. Function

Python provides function that can be abstracted into a function for easy program calls or provided to other modules.

7.1 Basic Use

```
def function_name(parameters):
    expressions
```

Python uses def to start the function definition, followed by the function name, inside the parentheses is the parameter of the function. And the internal code expressions aim to implement the specific function. If we want the function to have a return value, use return in expressions.

7.2 Example

```
def function():
    print('This is a function')
    a = 1+2
    print(a)
```

```
function()
```

```
This is a function
3
```

Above we defined a function named function, this function does not take parameters, so the inside of the parentheses is empty. The internal code is the body of the function. If we execute this script, we find that there is no output because we only defined the function and did not execute the function. At this point, we type function call function () into the Python command prompt, noting that the parenthesis for the function call cannot be omitted. Then the internal code inside the function will execute.

7.3 Function Parameter

When we call a function, we want to specify the value of some variables to be used in the function, then these variables are the parameters of the function. When the function is called, we need to pass in these parameters.

7.3.1 Basic Use

```
def function_name(parameters):
    expressions
```

7.3.2 Example

```
def func(a, b):
    c = a + b
    print('the c is ', c)
```

```
func()
```

```
TypeError                                 Traceback (most recent call last)
<ipython-input-27-bd1982955a12> in <module>
----> 1 func()

TypeError: func() missing 2 required positional arguments: 'a' and 'b'
```

```
func(2,3)
```

```
the c is  5
```

A function is defined here, and its parameters are two values, and its function is to add the two parameters together. If we do not specify the parameter func(), an error will occur. If we put a=2, b=3 into the function, and output the c is 5. Therefore, when calling a function, the number and position of parameters must be defined according to the function.

7.4 Default Parameters

When we define a function, some parameters are the same in most cases, but in order to improve the applicability of the function, we provide some alternative parameters. In order to facilitate the function call, we can set these parameters as the default parameters, so that the parameters need not be explicitly given in the function call process.

7.4.1 Basic Use

```
def function_name(para_1,...,para_n=defau_n,..., para_m=defau_m):
    expressions
```

When we define a function, we give some values to these default parameters. Pay attention that the default parameters can not appear before non-default parameters.

7.4.2 Example

```
def sale_car(price, color='red', brand='Ford', is_second_hand=True):
    print('price', price,
          'color', color,
          'brand', brand,
          'is_second_hand', is_second_hand)
```

```
sale_car(5000)
```

```
price 5000 color red brand Ford is_second_hand True
```

```
sale_car(5000,color='white')
```

```
price 5000 color white brand Ford is_second_hand True
```

Here we define a sale_car function that takes the attributes of the car, except the attribute price, the other attributes such as color, brand, and is_second_hand, all have default values. If we call sale_car(5000), it will have the same effect as sale_car(1000, 'red', 'Ford', True). It is also possible to pass in specific parameters during function calls to modify the default parameters. By default parameters we can reduce the complexity of our function calls.

8. Class

8.1 Define a Class

When we define a class, the first letter in a class name is recommended to be defined in the form of capital, such as a Calculator. The class can define its own attributes, for example the attribute name = 'Good Calculator'. In the class, we can also define some functions, such as def add (self, x, y), pay attention to the self here is a default value.

```
class Calculator:       # First letter should be capitalized, and the colon should not be missing
    name='Good Calculator'  # One attribute of this class
    price=18
    def add(self,x,y):
        print(self.name)
        result = x + y
        print(result)
    def minus(self,x,y):
        result=x-y
        print(result)
    def times(self,x,y):
        print(x*y)
    def divide(self,x,y):
        print(x/y)
```

```
cal=Calculator()  # Pay attention! Do not forget the () after the class name
```

```
cal.name
```

```
'Good Calculator'
```

```
cal.price
```

```
18
```

```
cal.add(10,20)
```

```
Good Calculator
30
```

```
cal.minus(20,10)
```

```
10
```

```
cal.times(2,3)
```

```
6
```

```
cal.divide(10,2)
```

```
5.0
```

8.2 init Function

__init__ function can initialize some values in a class.

```
class Calculator:
    name='good calculator'
    price=18
    # Pay attention,  the underscore here is a double underscore
    def __init__(self,name,price,height,width,weight):
        self.name=name
        self.price=price
        self.h=height
        self.wi=width
        self.we=weight
```

```
c=Calculator('bad calculator',18,17,16,15)
```

```
c.name
```

```
'bad calculator'
```

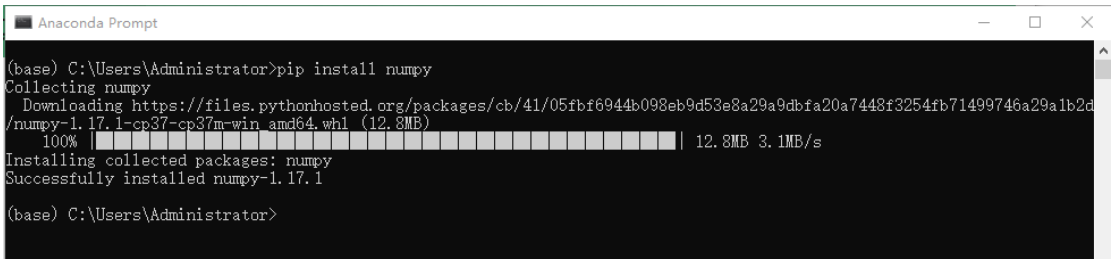
```
c.price,c.h,c.wi,c.we
```

```
(18, 17, 16, 15)
```

9. Package

9.1 Install Package

The easiest way to install packages is in Anaconda. We can use “pip“ in Anaconda Prompt to install a package.



9.2 Import Package

9.2.1 import time

```
import time
print(time.localtime())  # print local time
```

```
time.struct_time(tm_year=2019, tm_mon=8, tm_mday=30, tm_hour=10, tm_min=1, tm_sec=50, tm_wday=4, tm_yday=242, tm_isdst=1)
```

In this example, time is a package which already built-in Python. We can also install some other packages. For example, the package numpy need to be installed by ourselves.

9.2.2 import time as t

Sometimes the package name is very long, we can abbreviate it by this method.

```
import time as t
print(t.localtime()) # here we need to ust t instead of time anymore
```

```
time.struct_time(tm_year=2019, tm_mon=8, tm_mday=30, tm_hour=10, tm_min=9, tm_sec=12, tm_wday=4, tm_yday=242, tm_isdst=1)
```

9.2.3 from time import time, localtime

We can also just import the functions we need in the package.

```
from time import time, localtime
print(localtime())
print(time())
```

```
time.struct_time(tm_year=2019, tm_mon=8, tm_mday=30, tm_hour=10, tm_min=11, tm_sec=0, tm_wday=4, tm_yday=242, tm_isdst=1)
1567174260.6195483
```

9.2.4 from time import *

Using *, we can import all the functions in the package.

```
from time import *
print(localtime())
```

```
time.struct_time(tm_year=2019, tm_mon=8, tm_mday=30, tm_hour=10, tm_min=11, tm_sec=53, tm_wday=4, tm_yday=242, tm_isdst=1)
```

10. Read and Write Files

10.1 open

```
text = "This is a txt file."
```

```
my_file=open('my file.txt','w')   # Usage: open('file name','form') with 'w':write,'r': read.
my_file.write(text)               # This statement writes the previously defined text
my_file.close()                   # Close the file
```

```
file_content=open('my file.txt','r')
```

```
for line in file_content:
    print(line)
```

```
This is a txt file.
```

10.2 append

```
append_text='This is appended file.'
my_file=open('my file.txt','a')   # 'a'=append content into the file
my_file.write(append_text)
my_file.close()
```

```
file_content=open('my file.txt','r')
```

```
file_content=open('my file.txt','r')
for line in file_content:
    print(line)
```

```
This is a txt file.This is appended file.
```

```
my_file.close() # Do not forget to close the file after using it!
```

10.3 read

```
file= open('my file.txt','r')
content=file.read()
print(content)
```

```
This is a txt file.This is appended file.
```

11. Some Useful Packages

11.1 NumPy

NumPy (Numerical Python) is an extended library of Python language, which supports a large number of dimensional array and matrix operations and provides a large number of mathematical function libraries for array operations.

The official website: https://numpy.org/

10.2 Pandas

Pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.

The official website: https://pandas.pydata.org/

10.3 Matplotlib

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shells, the Jupyter notebook, web application servers, and four graphical user interface toolkits.

The official website: https://matplotlib.org/index.html

10.4 Scikit-learn

Scikit-learn is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

The official website: https://scikit-learn.org/stable/#