Tutorial 1: Introduction to Python

Reference: https://www.programiz.com/python-programming

This tutorial is created using the examples available on above reference website. The website provides online interactive tutorials.

History of Python

Python is a fairly old language created by Guido Van Rossum. The design began in the late 1980s and was first released in February 1991.

Why the name Python?

No. It wasn't named after a dangerous snake. Rossum was fan of a comedy series from late seventies. The name "Python" was adopted from the same series "Monty Python's Flying Circus".

Features of Python Programming

- **A simple language which is easier to learn**
  It's much easier to read and write Python programs compared to other languages like: C++, Java, C#.
- **Free and open-source**
- **Portability**
  You can move Python programs from one platform to another, and run it without any changes.
- **Extensible and Embeddable**
  Suppose an application requires high performance. You can easily combine pieces of C/C++ or other languages with Python code.
- **A high-level, interpreted language**
  It automatically converts your code to the language your computer understands. You don't need to worry about any lower-level operations.
- **Large standard libraries to solve common tasks**
  Python has a number of standard libraries which makes life of a programmer much easier since you don't have to write all the code yourself.
- **Object-oriented**
  Everything in Python is an object. Object oriented programming (OOP) helps you solve a complex problem intuitively.

Python in Linux

Python versions 2.x and 3.x are usually available in most modern Linux distributions out of the box. You can enter a Python shell by typing `python` or `python3` in your terminal and exit with `quit()`:

```
$ which python
$ which python3
$ python -v
$ python3 -v
$ python
>>> quit()
$ python3
>>> quit()
```

Your First Python Program

Create a directory called "*python_tutorial*" in your home directory. Inside that directory, create a new file called "*addnumbers.py*".

```
# Add two numbers
num1 = 3
num2 = 5
sum = num1+num2
print(sum)
```

How this program works?

Line 1: `# Add two numbers`

Any line starting with # in Python programming is a comment.

Line 2: `num1 = 3`

Here, *num1* is a variable. You can store a value in a variable. Here, 3 is stored in this variable.

Line 3: `num2 = 5`

Similarly, 5 is stored in *num2* variable.

Line 4: `sum = num1+num2`

The variables *num1* and *num2* are added using + operator. The result of addition is then stored in another variable *sum*.

Line 5: `print(sum)`

The *print()* function prints the output to the screen. In our case, it prints 8 on the screen.

Running Python scripts (for this tutorial, use the method number 2, script mode)

1. **Immediate mode:** You type the command at the Python console, one by one, and the answer is immediate.

Python console can get started by issuing the command:

```
$ python
>>> <type commands here>
```

>>> symbol is the Python prompt.

2. **Script mode**: On most of the UNIX systems, you can run Python scripts from the command line in the following manner

```
$ cd ~/python_tutorial
$ python addnumbers.py
```

3. Integrated Development Environment (**IDE**)

Using an IDE can get rid of redundant tasks and significantly decrease the time required for application development.

IDE is a graphical user interface (GUI) that can be installed along with the Python programming language and is available from the official website.

Possible issues

It is important to remember that Python takes tabs very seriously – so if you are receiving any error that mentions tabs, correct the tab spacing. Both tabs and spaces are supported, but the standard indentation requires standard Python code to use four spaces.

Python Keywords

Keywords are the reserved words in Python.

We cannot use a keyword as variable name, function name or any other identifier. They are used to define the syntax and structure of the Python language.

In Python, keywords are case sensitive.

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

Python Indentation

Indentation can be ignored in line continuation. But it's a good idea to always indent. It makes the code more readable. For example:

```
if True:
    print('Hello')
    a = 5
```

and

```
if True: print('Hello'); a = 5
```

both are valid and do the same thing. But the former style is clearer.

Incorrect indentation will result into IndentationError.

Python Variables

- Create a Python file “variables_example.py”.

```
website = "Apple.com"

print(website)
```

- Change value of a variable

```
website = "Apple.com"

# assigning a new variable to website
website = "Programiz.com"

print(website)
```

- Assigning multiple values to multiple variables

```
a, b, c = 5, 3.2, "Hello"

print (a)
print (b)
print (c)
```

- assign the same value to multiple variables at once

```
x = y = z = "same"

print (x)
print (y)
print (z)
```

Assigning value to a constant in Python

In Python, constants are usually declared and assigned on a module. Here, the module means a new file containing variables, functions etc which is imported to main file. Inside the module, constants are written in all capital letters and underscores separating the words.

Create a “constant.py”

```
PI = 3.14
GRAVITY = 9.8
```

Create a “constant_example.py”

```
import constant

print(constant.PI)
print(constant.GRAVITY)
```

In the above program, we create a constant.py module file. Then, we assign the constant value to PI and GRAVITY. After that, we create a main.py file and import the constant module. Finally, we print the constant value.

Data types in Python

Every value in Python has a datatype. Since everything is an object in Python programming, data types are actually classes and variables are instance (object) of these classes.

There are various data types in Python. Some of the important types are listed below.

1.Python Numbers

Integers, floating point numbers and complex numbers falls under Python numbers category. They are defined as *int*, *float* and *complex* class in Python.

We can use the *type()* function to know which class a variable or a value belongs to and the *isinstance()* function to check if an object belongs to a particular class.

Create "datatypes_example1.py".

```
a = 5
print(a, "is of type", type(a))

a = 2.0
print(a, "is of type", type(a))

a = 1+2j
print(a, "is complex number?", isinstance(1+2j,complex))
```

2.Python List

List is an ordered sequence of items. It is one of the most used datatype in Python and is very flexible. All the items in a list do not need to be of the same type.

Declaring a list is pretty straight forward. Items separated by commas are enclosed within brackets [].

```
>>> a = [1, 2.2, 'python']
```

We can use the slicing operator [ ] to extract an item or a range of items from a list. Index starts form 0 in Python.

Create "datatypes_example2.py".

```
a = [5,10,15,20,25,30,35,40]

# a[2] = 15
print("a[2] = ", a[2])

# a[0:3] = [5, 10, 15]
print("a[0:3] = ", a[0:3])

# a[5:] = [30, 35, 40]
print("a[5:] = ", a[5:])
```

3.Python Tuple

Tuple is an ordered sequence of items same as list. The only difference is that tuples are immutable. Tuples once created cannot be modified.

Tuples are used to write-protect data and are usually faster than list as it cannot change dynamically.

It is defined within parentheses () where items are separated by commas.

```
>>> t = (5,'program', 1+3j)
```

We can use the slicing operator [] to extract items but we cannot change its value.

Create “datatypes_example3.py”.

```
t = (5,'program', 1+3j)

# t[1] = 'program'
print("t[1] = ", t[1])

# t[0:3] = (5, 'program', (1+3j))
print("t[0:3] = ", t[0:3])

# Generates error
# Tuples are immutable
t[0] = 10
```

4.Python Strings

String is sequence of Unicode characters. We can use single quotes or double quotes to represent strings. Multi-line strings can be denoted using triple quotes, ''' or """.

```
>>> s = "This is a string"
>>> s = '''a multiline
```

Like list and tuple, slicing operator [ ] can be used with string. Strings are immutable.

Create “datatypes_example4.py”.

```
s = 'Hello world!'

# s[4] = 'o'
print("s[4] = ", s[4])

# s[6:11] = 'world'
print("s[6:11] = ", s[6:11])

# Generates error
# Strings are immutable in Python
s[5] ='d'
```

5.Python Set

Set is an unordered collection of unique items. Set is defined by values separated by comma inside braces { }. Items in a set are not ordered.

Create “datatypes_example5.py”.

```
a = {5,2,3,1,4,4,4}

# printing set variable
print("a = ", a)
```

```
# data type of variable a
print(type(a))
```

6.Python Dictionary

Dictionary is an unordered collection of key-value pairs.

It is generally used when we have a huge amount of data. Dictionaries are optimized for retrieving data. We must know the key to retrieve the value.

In Python, dictionaries are defined within braces {} with each item being a pair in the form *key:value*. Key and value can be of any type.

```
>>> d = {1:'value','key':2}
>>> type(d)
<class 'dict'>
```

We use key to retrieve the respective value. But not the other way around.

Create "datatypes_example6.py".

```
d = {1:'value','key':2}
print(type(d))

print("d[1] = ", d[1]);

print("d['key'] = ", d['key']);

# Generates error
print("d[2] = ", d[2]);
```

Python Import

When our program grows bigger, it is a good idea to break it into different modules.

A module is a file containing Python definitions and statements. Python modules have a filename and end with the extension *.py*.

Definitions inside a module can be imported to another module or the interactive interpreter in Python. We use the *import* keyword to do this.

For example, we can import the math module by typing in *import math*.

```
import math
print(math.pi)
```

Now all the definitions inside math module are available in our scope. We can also import some specific attributes and functions only, using the from keyword. For example:

```
>>> from math import pi
>>> pi
3.141592653589793
```

While importing a module, Python looks at several places defined in sys.path. It is a list of directory locations.

```
>>> import sys
>>> sys.path
['',
 'C:\\Python33\\Lib\\idlelib',
 'C:\\Windows\\system32\\python33.zip',
 'C:\\Python33\\DLLs',
 'C:\\Python33\\lib',
 'C:\\Python33',
 'C:\\Python33\\lib\\site-packages']
```

We can add our own location to this list as well.

Comparison operators in Python

>	x > y
<	x < y
==	x == y
!=	x != y
>=	x >= y
<=	x <= y

Python if...else Statement

Create “if_else_example1.py”

```
# Program checks if the number is positive or negative
# And displays an appropriate message

num = 3

# Try these two variations as well.
# num = -5
# num = 0

if num >= 0:
    print("Positive or Zero")
else:
    print("Negative number")
```

Create “if_else_example2.py”

```
# In this program,
# we check if the number is positive or
# negative or zero and
# display an appropriate message

num = 3.4

# Try these two variations as well:
# num = 0
# num = -4.5
```

```
if num > 0:
    print("Positive number")
elif num == 0:
    print("Zero")
else:
    print("Negative number")
```

Create “if_else_example2.py”

```
# In this program, we input a number
# check if the number is positive or
# negative or zero and display
# an appropriate message
# This time we use nested if

num = float(input("Enter a number: "))
if num >= 0:
    if num == 0:
        print("Zero")
    else:
        print("Positive number")
else:
    print("Negative number")
```

Python for Loop

The for loop in Python is used to iterate over a sequence (list, tuple, string) or other iterable objects.

Create “for_loop_example1.py”

```
# Program to find the sum of all numbers stored in a list

# List of numbers
numbers = [6, 5, 3, 8, 4, 2, 5, 4, 11]

# variable to store the sum
sum = 0

# iterate over the list
for val in numbers:
      sum = sum+val

# Output: The sum is 48
print("The sum is", sum)
```

The range() function

We can generate a sequence of numbers using *range()* function. *range(10)* will generate numbers from 0 to 9 (10 numbers).

We can also define the start, stop and step size as *range(start,stop,step size)*. step size defaults to 1 if not provided.

The following example will clarify this.

```
# Output: range(0, 10)
print(range(10))

# Output: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
print(list(range(10)))

# Output: [2, 3, 4, 5, 6, 7]
print(list(range(2, 8)))

# Output: [2, 5, 8, 11, 14, 17]
print(list(range(2, 20, 3)))
```

We can use the range() function in for loops to iterate through a sequence of numbers. It can be combined with the len() function to iterate though a sequence using indexing. Here is an example.

Create “for_loop_example2.py”

```
# Program to iterate through a list using indexing

genre = ['pop', 'rock', 'jazz']

# iterate over the list using index
for i in range(len(genre)):
      print("I like", genre[i])
```

for loop with else

A for loop can have an optional else block as well. The else part is executed if the items in the sequence used in for loop exhausts.

Create “for_loop_example2.py”

```
digits = [0, 1, 5]

for i in digits:
    print(i)
else:
    print("No items left.")
```

Python while Loop

The while loop in Python is used to iterate over a block of code as long as the test expression (condition) is true.

Create “while_loop_example1.py”

```
# Program to add natural
# numbers upto
# sum = 1+2+3+...+n

# To take input from the user,
# n = int(input("Enter n: "))

n = 10
```

```
# initialize sum and counter
sum = 0
i = 1

while i <= n:
    sum = sum + i
    i = i+1    # update counter

# print the sum
print("The sum is", sum)
```

while loop with else

Same as that of for loop, we can have an optional else block with while loop as well.

Create “while_loop_example2.py”

```
# Example to illustrate
# the use of else statement
# with the while loop

counter = 0

while counter < 3:
    print("Inside loop")
    counter = counter + 1
else:
    print("Inside else")
```

Python break statement

The break statement terminates the loop containing it. Control of the program flows to the statement immediately after the body of the loop.

Create “break_stat_example.py”

```
# Use of break statement inside loop

for val in "string":
    if val == "i":
        break
    print(val)

print("The end")
```

Python continue statement

The continue statement is used to skip the rest of the code inside a loop for the current iteration only. Loop does not terminate but continues on with the next iteration.

Create “continue_stat_example.py”

```
# Program to show the use of continue statement inside loops

for val in "string":
    if val == "i":
        continue
    print(val)

print("The end")
```

Python pass statement

In Python programming, pass is a null statement. The difference between a comment and pass statement in Python is that, while the interpreter ignores a comment entirely, pass is not ignored.

However, nothing happens when pass is executed. It results into no operation (NOP).

Create “pass_stat_example.py”

```
# pass is just a placeholder for
# functionality to be added later.
sequence = {'p', 'a', 's', 's'}
for val in sequence:
    pass
```

Python Looping Techniques

1.The infinite loop

We can create an infinite loop using while statement. If the condition of while loop is always True, we get an infinite loop.

Create "loop_example1.py".

```
# An example of infinite loop
# press Ctrl + c to exit from the loop

while True:
   num = int(input("Enter an integer: "))
   print("The double of",num,"is",2 * num)
```

2.Loop with condition at the top

Create "loop_example2.py".

```
# Program to illustrate a loop with condition at the top

# Try different numbers
n = 10

# Uncomment to get user input
#n = int(input("Enter n: "))

# initialize sum and counter
sum = 0
i = 1
```

```
while i <= n:
    sum = sum + i
    i = i+1    # update counter

# print the sum
print("The sum is",sum)
```

3.Loop with condition in the middle

Create "loop_example3.py".

```
# Program to illustrate a loop with condition in the middle.
# Take input from the user until a vowel is entered

vowels = "aeiouAEIOU"

# infinite loop
while True:
    v = input("Enter a vowel: ")
    # condition in the middle
    if v in vowels:
        break
    print("That is not a vowel. Try again!")

print("Thank you!")
```

4.Loop with condition at the bottom

Create "loop_example4.py".

```
# Python program to illustrate a loop with condition at the bottom
# Roll a dice until user chooses to exit

# import random module
import random

while True:
    input("Press enter to roll the dice")

    # get a number between 1 to 6
    num = random.randint(1,6)
    print("You got",num)
    option = input("Roll again?(y/n) ")

    # condition
    if option == 'n':
        break
```

Python Functions

Once we have defined a function, we can call it from another function, program or even the Python prompt. To call a function we simply type the function name with appropriate parameters.

Create "function_example1.py".

```
def absolute_value(num):
      """This function returns the absolute
      value of the entered number"""

      if num >= 0:
            return num
      else:
            return -num

# Output: 2
print(absolute_value(2))

# Output: 4
print(absolute_value(-4))
```

Scope and Lifetime of variables

Scope of a variable is the portion of a program where the variable is recognized. Parameters and variables defined inside a function is not visible from outside. Hence, they have a local scope.

Create "function_example2.py".

```
def my_func():
      x = 10
      print("Value inside function:",x)

x = 20
my_func()
print("Value outside function:",x)
```

Create "function_example3.py".

```
def greet(name,msg):
   """This function greets to
   the person with the provided message"""
   print("Hello",name + ', ' + msg)

greet("Monica","Good morning!")
```

Create "function_example3.py".

```
def greet(name, msg = "Good morning!"):
   """
   This function greets to
   the person with the
   provided message.

   If message is not provided,
   it defaults to "Good
   morning!"
   """

   print("Hello",name + ', ' + msg)

greet("Kate")
greet("Bruce","How do you do?")
```

Using Global and Local variables in same code

Create "global_local_example.py".

```
x = "global"

def foo():
    global x
    y = "local"
    x = x * 2
    print(x)
    print(y)

foo()
```

NumPy Array

NumPy is a package for scientific computing which has support for a powerful N-dimensional array object.

NumPy provides multidimensional array of numbers (which is actually an object). Let's take an example:

Create "numpy_example1.py".

```
import numpy as np
a = np.array([1, 2, 3])
print(a)                # Output: [1, 2, 3]
print(type(a))          # Output: <class 'numpy.ndarray'>
```

As you can see, NumPy's array class is called ndarray.

Addition of Two Matrices

We use + operator to add corresponding elements of two NumPy matrices.

Create "numpy_example1.py".

```
import numpy as np

A = np.array([[2, 4], [5, -6]])
B = np.array([[9, -3], [3, 6]])
C = A + B      # element wise addition
print(C)
```