Python Tutorial

Prof. Ankur Sinha

Indian Institute of Managemet Ahmedabad

Gujarat India

About Python

- Python was created by Guido van Rossum and first released in 1991. It is a powerful language with efficient high-level data structures that allow easy programming. It is a useful language for rapid development of applications.
- Python is an interpreted, high-level, general-purpose programming language that features a dynamic type system and automatic memory management. It supports multiple programming paradigms, like procedural and functional. Its popularity for data science is driven by the ease of writing code and the availability of a comprehensive library.
- Python interpreters are freely available for most of the commonly used operating systems. CPython, the reference implementation of Python, is open source software.

Reference: https://en.wikipedia.org/wiki/Python_(programming_language)

- Interpreted: No need to compile (converting the script into machine readable form) a python script before executing it. The interpreter executes the script directly hiding the low level tasks from the user.

- High-level: Strong abstraction from the details of the computer. The user need not know intricate functioning of the computer to write their code.

- General-purpose: Can be applied to wide variety of application domains. The constructs do not restrict python in anyway so that it cannot be used for any particular application.

- Dynamic type system: The variables, expressions, functions or modules need not be assigned a type in python. Python is able to take care of the types internally and dynamically during run time. The opposite of a Dynamic type system is a Static type system.

- Object-oriented: Allows variables (attributes) and functions (methods) to be bundled into an object.

- Procedural programming: It is a type of imperative programming which contains a list of commands bundled as a procedure.

- Functional programming: It is designed on the concept of mathematical functions and does not support state. The functions in functional programming are pure functions as you define functions in Maths.

Python Indentations

- Python uses indentation to indicate a block of code

  Example

```
a=5
b=2
if a > b:
    print("a is greater than b!")
```

- Python will give you an error if you skip the indentation:

  Example

```
a=5
b=2
if a > b:
print("a is greater than b!")
```

  <Error>

Creating Variables

- Python does not require declaration of variables
- As soon as you assign a value to a variable, it gets created

  Example

```
name = "Anuj"
age = 36
print(name,age)
```

- Python automatically recognizes the data type based on the value that is assigned to a variable

  Example

```
x = 5                 # x is of type int
x = "Hello"           # x is now of type str
print(x)              # This will print "Hello" as variable x has been assigned a new value
```

Python Numbers

- There are three numeric types in Python:
  - int
  - float
  - complex
- Variables of numeric types are created automatically as soon as you assign a value to them:

  Example

```
x = 23          # int
y = 5.6         # float
z = 1+2j        # complex
```

- To identify the type of any variable or object in python, use the type() function

Python Operators

- Operators are used to perform different kinds of operations on variables and values
- Following is a list of operators in Python:
  - Arithmetic operators (+,-,*,/,//,%,**)
  - Assignment operators (=,+=,-= etc)
  - Comparison operators (==,!=,>,<,>=,<=)
  - Logical operators (and, or, not)
  - Membership operators (in, not in)
  - Identity operators (is, not is)
  - Bitwise operators (works on bits)

Let us make a calculator

Python Strings

- String literals can be defined in Python either using single quotation marks, or double quotation marks
- You may define a string in either of the two ways: 'Hello World' or "Hello World"
- Square brackets are used to access elements of the string

  Example

```
a = "Hello, World!"
print(a[1])                #Prints: "e"
```

- String slicing can be done using the square brackets

  Example

```
b = "Hello, World!"
print(b[2:5])              #Prints: "llo"
```

- Strings can be concatenated using the + sign and can be repeated using the * sign

Let us try some string operations

About Printing

- Printing arguments

  Example

```
name = "Anuj"
age = 36
print(name, age)                    #Prints: Anuj 36
```

- F-strings provide a convenient way to embed Python expressions inside strings

  Example

```
name = "Anuj"
age = 36
print(f"{name} is {age} years old")    #Prints: Anuj is 36 years old
```

Python Casting

- Type casting is commonly used to convert one data type into another data type
- Casting in Python is done using constructor functions as shown below:
  - int() - converts float or string to integer
  - float() - converts integer or string to float
  - str() - converts integer or float to string

Python Casting

Examples

- Integers:

```
y = int(3.2)          # y will be 3
z = int("3")          # z will be 3
```

- Floats:

```
x = float(2)          # x will be 2.0
z = float("3")        # z will be 3.0
w = float("1.2")      # w will be 1.2
```

- Strings:

```
y = str(20)           # y will be "20"
z = str(3.0)          # z will be "3.0"
```

Python Data Types

There are four commonly used data types in the Python programming language:

- **List**
  - Ordered and changeable
  - Allows duplicate members
- **Tuple**
  - Ordered and unchangeable
  - Allows duplicate members
- **Set**
  - Unordered and unindexed
  - No duplicate members
- **Dictionary**
  - Unordered, changeable and indexed
  - No duplicate members

Python List

- Ordered and changeable
- Lists are written with square brackets

Example

```
mylist = ["grapes", "banana", "mango"]
print(mylist)

mymixedlist = ["grapes", 100, "banana", 12, "mango", 5.0]
print(mymixedlist)

list1 = ["Once","upon","a","time"]
list2 = ["there","was","a","king"]
list3 = list1+list2
print(list3)          #Prints: ["Once", "upon", "a", "time", "there", "was", "a", "king"]
print(list3[::-1])    #Prints: ["king", "a", "was", "there", "time", "a", "upon", "Once"]
```

Python Tuples

- Ordered and unchangeable
- Tuples are written with round brackets

Example

```
mytuple = ("grapes", "banana", "mango")
print(mytuple)
```

Python Sets

- Unordered and unindexed
- Sets are written with curly brackets

Example

```
myset = {"grapes", "banana", "mango"}
print(myset)
```

```
#Access items in set
myset = {"grapes", "banana", "mango"}
for x in myset:
    print(x)
```

```
#Check if an item is present in a set
myset = {"grapes", "banana", "mango"}
print("banana" in myset)
```

Python Dictionaries

- Unordered, changeable and indexed
- Dictionaries are written with curly brackets
- Dictionaries have keys and values.

Example

```
mydict = {
  "first name": "Anuj",
  "last name": "Gupta",
  "age": 23
}
print(mydict)
```

Python If ... Else

- Python supports the following logical conditions from mathematics

  Equals: a == b

  Not Equals: a != b

  Less than: a < b

  Less than or equal to: a <= b

  Greater than: a > b

  Greater than or equal to: a >= b

- The logical conditions return a value True or False

  Example

```
a = 25
b = 33
print(a>b)                #Prints: False
```

Python If ... Else

- The "if statement" executes the block of commands under it, if the logical condition is true

Example

```
a = 25
b = 100
if b > a:
    print("b is greater than a")
```

Python If ... Else

- If there are multiple conditions to be checked then "elif keyword" is helpful
- The "else keyword" catches anything which is not caught by the "if keyword" or the "elif keyword"

Example

```
a = 100
b = 25
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")
```

Python Loops

- Python has two primitive loop commands:
  - while loop
  - for loop

Python While Loop

- A while loop executes a set of statements as long as a condition is true

Example

```
i = 1
while i < 6:
  print(i)
  i += 1
```

Python For Loop

- A for loop is used for iterating over a string, list, tuple, set or dictionary

Example

```
fruits = ["grapes", "banana", "mango"]
for x in fruits:
    print(x)
```

Print the length of words in a list

- lst = ["grapes", "banana", "mango"]

Print Fibonacci series

range() function

- It is a built in function that returns range object that is a sequence of integers
- Its syntax is range (start, stop[, step])
- Print even numbers in reverse order from 100

  Example

```
for i in range(100,0,-2):
    print(i,end=", ")
```

break and continue statements

- break statement breaks the innermost enclosing for or while loop
- continue statement continues with the next iteration of the for or while loop

Python Functions

- A function is a block of code that is executed when the function name is called
- A function may take inputs that are called arguments or parameters
- A function may return one or more variables as a tuple

  Example

```
def my_function():
    print("Hello World")
```

- To call the above function use the following command

```
my_function()
```

Python Functions

Example

```
def my_function(a,b):
    c = a+b
    d = a-b
    return c,d

add, sub = my_function(15,10)
print(add)              #Prints 25
print(sub)              #Prints 5

t = my_function(15,10)
print(t)                        #Prints: (25,5) note that "t" is a tuple
```

Print prime numbers up to 100

Python Lambda

- Python facilitates lambda function, which is a small anonymous function

Example

```
x = lambda a : a + 10
print(x(5))
```

Example

```
x = lambda a, b : a * b
print(x(5, 6))
```

Python Iterators

- An iterator is an object that contains a countable number of values and can be iterated upon
- Lists, tuples, sets and dictionaries are all iterable objects
- A string can also be iterated over the sequence of characters that it contains

Example

```
mytuple = ("grapes", "banana", "mango")

myit = iter(mytuple)

print(next(myit))
print(next(myit))
print(next(myit))
```

Python Classes/Objects

- A Class is like an object constructor
- Almost everything in Python is an object, with its attributes and methods

- To create a class, use the keyword class:

  Example

```
class Rectangle:
    length = 15
    breadth = 19
```

  Example

```
p1 = Rectangle()
print(p1.length)
print(p1.breadth)
```

Python Classes/Objects

Reference to the current instance of the class

Example

```
class Rectangle:
    def __init__(self,l,b):
        self.length = l
        self.bredth = b

    def area(self):
        return(self.length*self.bredth)

    def perimeter(self):
        return(2*self.length+2*self.bredth)
```

Constructor gets executed when the class is initiated

An instance of the class gets created

```
r1 = Rectangle(5,6)
r1.area()          #Prints: 30
r1.perimeter()     #Prints: 22
```

Python File Handling

- File Handling
  - The key function for working with files in Python is the open() function.
  - The open() function takes two parameters; *filename*, and *mode*.
  - There are four different methods (modes) for opening a file
    - "r" - Read - Default value. Opens a file for reading, error if the file does not exist
    - "a" - Append - Opens a file for appending, creates the file if it does not exist
    - "w" - Write - Opens a file for writing, creates the file if it does not exist
    - "x" - Create - Creates the specified file, returns an error if the file exists

Open a File

- To open the file, use the open() function
- The open() function returns a file object
- The returned object has a read() method for reading the content of the file

Example

```
f = open("myfile.txt", "r")      #Opens the file
print(f.read())                  #Reads the entire content of the file
f.close()                        #Closes the file
```

Read a file

- The read() method reads the entire file
- Read(n) reads n characters of the file

Example

```
f = open("myfile.txt", "r")
print(f.read(5))          #Prints the first 5 characters
f.seek(0)                 #Takes you to beginning of file (0 denotes 0 byte)
print(f.read())           #Prints the entire file from beginning
f.close()
```

Write a File

- To write to an existing file, add one of the following parameters to the open() function:
  - "a" - Append - will add content to the end of the file
  - "w" - Write - will overwrite the existing file

Example

```
f = open("myfile.txt", "a")
f.write("Appending content to the file.")
f.close()
```

Python Delete File

- To delete a file, you must import the OS module, and run its os.remove() function.
- Remove the file "myfile.txt":

Example

```
import os
os.remove("myfile.txt")
```

Examples

I will create a calculator using Python
The calculator will do the following tasks
* Addition
* Subtraction
* Multiplication
* Division
* Power

```
a = 100
b = 200
operation = "Addition"
```

```
if operation=="Addition":
    print(a+b)
if operation=="Subtraction":
    print(a-b)
if operation=="Multiplication":
    print(a-b)
if operation=="Division":
    print(a-b)
if operation=="Power":
    print(a-b)
```

```
300
```

I will create a calculator using Python

The calculator will do the following tasks

- Addition
- Subtraction
- Multiplication
- Division
- Power

```
a = 100
b = 200
operation = "Addition"
```

```
if operation=="Addition":
    print(a+b)
if operation=="Subtraction":
    print(a-b)
if operation=="Multiplication":
    print(a-b)
if operation=="Division":
    print(a-b)
if operation=="Power":
    print(a-b)
```

```
300
```

Fibonacci series

```
a,b = 0,1
while a<100:
        print(a)
        a,b = b,a+b
```

Print the length of words in a list

```
lst = ["grapes","banana","mango"]
for w in lst:
    print(w,len(w))
```

Prime numbers up to 100

```
def isPrime(n):
    for i in range(2,int(n**.5)+1):
        print(i)
        if n%i==0:
            return(False)
    return(True)


for i in range(2,101):
    if(isPrime(i)):
        print(i, end=", ")
```