**Python Basic 101**

Part 1: Getting Started

**2nd Edition**

Created By

Nhan Tran | Angela Cao

*April 27, 2023*

**Table of Contents**

[Chapter 1 – Python Basic 2](#_Toc131422542)

[Python Facts 2](#_Toc131422543)

[About: Python Fundamental 101 2](#_Toc131422544)

[Reserved Words 3](#_Toc131422545)

[Basic Terminologies: 3](#_Toc131422546)

[Loop 4](#_Toc131422547)

[Conditionals (if/elif/else): 4](#_Toc131422548)

[Chapter 2 – Python Objects 6](#_Toc131422549)

[Built-in Data Types 6](#_Toc131422550)

[Immutable Objects 6](#_Toc131422551)

[Mutable Objects 7](#_Toc131422552)

[Difference between: Immutable and Mutable 7](#_Toc131422553)

[Exceptions: Immutable and Mutable rule 7](#_Toc131422554)

[How To Access An Element In A... 8](#_Toc131422555)

[Class Objects: 8](#_Toc131422556)

[Functions vs Methods: 9](#_Toc131422557)

[Lambdas: 9](#_Toc131422558)

[Iterators: 9](#_Toc131422559)

[List Methods 9](#_Toc131422560)

[Chapter 3 – Operators 11](#_Toc131422561)

[Arithmetic Operators 11](#_Toc131422562)

[Assignment Operators 11](#_Toc131422563)

[Comparison Operators 12](#_Toc131422564)

[Logical Operators 12](#_Toc131422565)

[Identity Operators 12](#_Toc131422566)

[Membership Operators 12](#_Toc131422567)

[Bitwise Operators 12](#_Toc131422568)

[Escape Characters 13](#_Toc131422569)

[RegEx Library 13](#_Toc131422570)

[RegEx Functions 13](#_Toc131422571)

[RegEx Metacharacters 14](#_Toc131422572)

[RegEx Special Sequences 14](#_Toc131422573)

[References 16](#_Toc131422574)

# Chapter 1 – Python Basic

## Python Facts

* Python is an **interpreted**, **high-level**, **general-purpose** programming language.
  + **Interpreted**: the scripts/commands is executed one line at a time by an interpreter (Shell)
  + **High-level:** scripts are readable by human vs. being machine-friendly (instruction in 0 & 1)
  + **General-purpose:** the program language can accomplish just about anything.
* Python is an open source language and freely available for the programmers.
* Python supports all the major platforms such as windows. Linux, Macintosh. It is a cross-platform and highly portable language.
* User-Friendly, enables developer to write fewer codes for writing programs.
* Python is an object-oriented programming language.
* Everything in Python is an object. E.g., dataframe, list, etc.

## About: Python Fundamental 101[[1]](#endnote-1)

* Python is Case sensitive
* Blocks of code are denoted by line indentation, which is rigidly enforced.
* Whitespace indentation represent line-break for a command or denote blocks between parent and child code. Example:

im\_a\_parent:

∙∙∙∙im\_a\_child:

∙∙∙∙∙∙∙∙im\_a\_grandchild

∙∙∙∙im\_another\_child:

∙∙∙∙∙∙∙∙im\_another\_grand\_child

* A variable name cannot start with a number. E.g., **var2** **NOT** **2var**
* Only alphanumeric characters (A-z, 0-9) and underscores (\_) are allowed within a variable name. E.g., **memorialHermann\_97** **NOT** **memorial Hermann;**
* White space between objects are optional
* To output a Python object on the screen or terminal, you must call the print function with the Python object to be printed in between the parentheses of the function. Example:

>>> print(“Hello World”)

**Output**: “Hello World”

* Multi-Line Statements uses (\) or indentation for statements within the [], {}, or () brackets
* Semicolon (;) allows multiple statements on the single line given that neither statement starts a new code block.
* Python accepts single ('), double (") and triple (''' or """) quotes to denote string, as long as the same type of quote starts and ends the string.
* The triple quotes are used to span the string across multiple lines.
* A hash sign (#) that is not inside a quote begins a comment.
* A group of individual statements, which make a single code block are called suites in Python.

if x == y :

print('something')

elif expression :

suite

else :

suite

* Python allows you to assign a single value to several variables simultaneously. a = b = c = 1 or multiple variables in one line a,b,c = 1,2,"john"
* A single equal sign (‘=’) is an assignment operator or an operator that assigns a value to an element while a double equal sign (‘==’) is a comparison operator that checks whether two elements or values are equal to each other.

## Reserved Words

The following list are reserved words and cannot be used as a constant or variable name. All of Python keywords are in lowercase letters only.

|  |  |  |
| --- | --- | --- |
| and | exec | not |
| assert | **finally** | **or** |
| break | **for** | **pass** |
| class | **from** | **print** |
| continue | **global** | **raise** |
| def | **if** | **return** |
| del | **import** | **try** |
| elif | **in** | **while** |
| else | **is** | **with** |
| except | **lambda** | **yield** |

## Basic Terminologies:

**Variable**: is a word that stores a value. X = 10 x is the variable and 10 is the value

**Function**: a snippet of codes that allow user to reuse the code without having to rewrite it. This also help to organize the code, reduce writing, and improve readability. Example: min()

* There are 2 types of functions: built-in (dir()) and user-defined functions (myFunction())
* To write a function in Python, it must be declared using the “def” keyword. Infinite amount of arguments can be passed into the functions as necessary information. To return a value from the function, the “return” keyword is used. Example:

def my\_function(arg1, arg2, arg3):

∙∙∙∙sum1 = arg1 + arg2

∙∙∙∙result = sum1 \* arg3

∙∙∙∙return result

**Module**: a file containing Python definitions and statements. The file name is the module name with the suffix .py appended.

**Package/ Library:** a collection of functions, methods, and module(s) that serve a particular use purpose. It is also a way of structuring Python’s module namespace by using “dotted module names”. For example, the module name ***Pandas***.***DataFrame*** designates ***Pandas*** as the package and ***DataFrame*** as the submodule. **Package** contain **modules** and each **module** contains **functions**.

## Loop

|  |  |
| --- | --- |
| **Types of Loop** | **Description** |
| **While loop**  while x == b: | Repeats a statement or group of statements while a given condition is TRUE. It tests the condition before executing the loop body. |
| **For loop**  for x in range(10): | Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable. |
| **In-line Loop**  [x for x in range(10)]  {x:y for x,y in dict.items()} | A one-line commands of loop(s) within a [] or {}, and is used for creating a list or a dictionary comprehension object.  https://www.w3schools.com/python/python\_lists\_comprehension.asp |

*Loops can also be nested within each other.*

|  |  |
| --- | --- |
| **Loop Control Statement** | **Descriptions** |
| **Break statement** | Terminates the loop statement and transfers execution to the statement immediately following the loop. |
| **Continue statement** | Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating. |
| **Pass statement** | The pass statement in Python is used when a statement is required syntactically but you do not want any command or code to execute. |

### Conditionals (if/elif/else):

The Python conditional statement consists of these parts:

* **if** [required]: The “if” statement evaluates the primary conditional statement(s) and executes the indented block below the “if” statement if the conditional statement(s) is/are true. The “if” statement can be used multiple times, but with different conditional statements.
* **else**: If none of the conditional statement(s) expressed in the “if” or even “elif” statement is/are true, then the indented block of code below the “else” statement executes. Since there is no specific condition attached to this statement, the “else” statement can only be declared once.
* **elif**: The “elif” statement can be used to evaluate other conditional statement(s) if the conditional statement in the “if” statement fails. If the conditional statement in the “if” statement fails, but passes the conditional statement in the “elif” statement, then the indented block of code below the “elif” statement executes. The “elif” statement is the exact equivalent of the “else if” statement used in other programming languages for conditionals. The “elif” statement can be used multiple times, but with different conditional statements.

Example:

if x == y :

print('something')

elif expression :

suite

else :

suite

# Chapter 2 – Python Objects[[2]](#endnote-2)[[3]](#endnote-3)

## Built-in Data Types [[4]](#endnote-4)

In programming, data type is an important concept.

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:

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Type:** | **Name** | **Example** |
| Base types | Text | str | 'hello world' |
| Numeric | int | 120 |
| float | 1.21 |
| complex | 2+3j |
| Binary | byte |  |
| Boolen | bool | True/False |
| Collection types | Sequences | list | [1, 'one', 2, 'two'] |
| tuple | (1, 'one', 2, 'two') |
| range | range(1,10) -> (1,2,3,4,5,6,7,8,9,10) |
| Set | set | {'jack', 'sjoerd'} |
| Mapping | dict | {'jack': 4098, 'sjoerd': 4127} |
| Data structure types | numpy | array | 1-dimensional structure storing a list of the same data type. |
| pandas | series | like an array but each value has an index associated. |
| dataframe | 2-dimensional labeled/indexed data structure. In another word, dataframe = multiple series/array. |

## Immutable Objects

Doesn’t allow modification after creation

|  |  |
| --- | --- |
| **Integer (int):** 1, 20, 352 | Whole number value s |
| **Decimal Number (float):** 1.1, 4.589 | Decimal number values |
| **Boolean (bool):** True/False, 0/1 | Binary value |
| **Character String (str):** 'ABC', 'abc' | Alphabet & non-numeric values, presented in " " or ' ' |
| **Tuple:** ('physi', 2000) | A sequence of items in (), values are ordered, duplicated values allow |
| **Unicode:** 0061 -> a | Characters are presented by codes |

## Mutable Objects

Allow modification after creation

|  |  |
| --- | --- |
| **List:**  ['Lane', "home2", 23, 2.56] | A sequence (ordered) of objects in [], items can be different type, duplicated values allow |
| **Dictionary (dict):**  {'two':"one", 'fruit': 'app'} | Unordered data represented by key: value pairs in { }. No duplicate key allows. Only hold a single item for Key & Value |
| **Set:**  {"apple", "banana", "cherry"} | A collection of objects in { } which is unordered and unindexed. No duplicate values allow. |
| **Array:**  numpy.array([1, 4, 2, 5, 3]) | A collection of same-type items stored together for fast processing. Allow random access of elements |

### Difference between: Immutable and Mutable

* Immutable are quicker to access than mutable objects.
* Mutable objects are great to use when you need to change the size of the object, example list, dict etc. Immutables are used when you need to ensure that the object you made will always stay the same.
* Immutable objects are fundamentally expensive to “change”, because doing so involves creating a copy and this take up computer resources (memory). Changing mutable objects is cheap.

|  |  |
| --- | --- |
| **Mutable** | **Immutable** |
| When called and change is made to this object, the **change will be store back** into the **original variable**. To keep the original object unchanged a copy need to be made prior to the change. | When called and change is made to this object, the **change will be lost if not store in a variable.** To save the change assigned the change to a variable. |
| N = [5, 6]  N += 10 -> [5, 6, 10], which is also the new value of N | N = 5  N += 5 -> 10, but N will still be 5 |

### Exceptions: Immutable and Mutable rule

The key thing to note is that the ***bindings*** are unchangeable, not the objects they are bound to.

T = (‘holberton’, [1, 2, 3])

The above tuple **t** contains elements of different data types, the first one is an immutable string and the second one is a mutable list. The tuple itself isn’t mutable. i.e., it doesn’t have any methods for changing its contents. Likewise, the string is immutable because strings don’t have any mutating methods. But the list object does have mutating methods, so it can be changed. This is a subtle point, but nonetheless important: the “value” of an immutable object ***can’t*** change, but its constituent objects ***can***.[[5]](#endnote-5)

## How To Access An Element In A...

Allow modification after creation

|  |  |
| --- | --- |
| **List:**  x = ['Lane', "home2", 23]  print(x[0]) | To access an element in a list, refer by the index number of the belonging list (0 is the index of the first element, 1 is the index of the second element, etc.) |
| **Dictionary (dict):**  x = {'two':"one", 'fruit': 'app'}  print(x[‘two’]) | To access an element in a dictionary, refer by the key of the element. |
| **Set:**  x = {"apple", "banana", "cherry"}  print(x[0]) | To access an element in a set, refer by the index number of the belonging set (0 is the index of the first element, 1 is the index of the second element, etc.) |
| **Tuple:**  x =('physi', 2000)  print(x[0]) | To access an element in a tuple, refer by the index number of the belonging tuple (0 is the index of the first element, 1 is the index of the second element, etc.) |
| **Array:**  x = numpy.array([1, 4, 2, 5])  print(x[0]) | To access an element in an array, refer by the index number of the belonging array (0 is the index of the first element, 1 is the index of the second element, etc.) |

### Class Objects:

A Class is like an object constructor, or a “blueprint” for creating objects.

* Attribute references: store reference information. E.g., MyClass.i
* Instantiation: it look like a function and is used to create new empty object. All classes have a function called \_\_init\_\_(), which is always executed when the class is being initiated.
* The class object is declared with the “class” keyword.

Example:

class exampleClass:

def \_\_init\_\_(self, param):

self.param = param

def sampleFunc(self):

print(self.param)

c = exampleClass(“Hello”)

c.sampleFunc()

### Functions vs Methods:[[6]](#endnote-6)

Functions and Methods are almost identical to each other as they perform in a very similar way, but the crucial difference between the two is that while functions can be called independently or by themselves by their own name, methods cannot be called by their names only, and must also include the class or object that the specific method belongs to.

**Examples:**

* Function: abs(-25)
* Method: math.floor(6.49)

### Lambdas: [[7]](#endnote-7)

Lambda is a small anonymous function that involves an infinite number of arguments, but one expression. Lambdas are declared using the “lambda” keyword.

Syntax: lambda *arguments* : *expression*

Example: lambda a, b : a + b

### Iterators: [[8]](#endnote-8)

An iterator is an object that contains a countable number of values or that can be iterated upon.

* An iterator can be declared with the iter() function with the object inside the parentheses of the function.
* To call on the next item in an iterator, call the next() function with the iterator inside.

## List Methods[[9]](#endnote-9)

|  |  |
| --- | --- |
| Method | Description |
| append(x) | Adds an element **x** at the end of the list |
| clear() | Removes all the elements from the list |
| copy() | Returns a copy of the list |
| count(x) | Returns the number of elements with the specified value **x** |
| extend(s) | Add the elements of a list (or any iterable) **s**, to the end of the current list |
| index(x) | Returns the index of the first element with the specified value **x** |
| insert(x, y) | Adds an element y at the specified position **x** |
| pop(x) | Removes the element at the specified position **x** |
| remove(x) | Removes the item with the specified value **x** |
| reverse() | Reverses the order of the list |
| sort([reverse], [key]) | Sorts the list (optional parameters: reverse-True sorts the list descending order; key specifies sorting criteria(s)) |

# Chapter 3 – Operators[[10]](#endnote-10)

Operators are used to perform operations on variables and values.

Python divides the operators in the following groups:

* **Arithmetic operators**: Used with numeric values to perform common mathematical operations
* **Assignment operators**: used to assign values to variable
* **Comparison operators**: used to compare two values
* **Logical operators**: Used to combine conditional statements
* **Identity operators**: Used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location
* **Membership operators**: Used to test if a sequence is presented in an object
* **Bitwise operators**: Used to compare (binary) numbers

## Arithmetic Operators

|  |  |
| --- | --- |
| Operator | Name |
| + | Addition |
| - | Subtraction |
| \* | Multiplication |
| / | Division |
| % | Modulus (return the remainder of the division) |
| \*\* | Exponentiation |
| // | Floor division (round down the product) |

## Assignment Operators

|  |  |  |
| --- | --- | --- |
| Operator | Example | Same As |
| = | x = 5 | x = 5 |
| += | x += 3 | x = x + 3 |
| -= | x -= 3 | x = x – 3 |
| \*= | x \*= 3 | x = x \* 3 |
| /= | x /= 3 | x = x / 3 |
| %= | x %= 3 | x = x % 3 |
| //= | x //= 3 | x = x // 3 |
| \*\*= | x \*\*= 3 | x = x \*\* 3 |
| &= | x &= 3 | x = x & 3 |
| |= | x |= 3 | x = x | 3 |
| ^= | x ^= 3 | x = x ^ 3 |
| >>= | x >>= 3 | x = x >> 3 |
| <<= | x <<= 3 | x = x << 3 |

## Comparison Operators

|  |  |  |
| --- | --- | --- |
| Operator | Name | Example |
| == | Equal | x == y |
| != | Not equal | x != y |
| > | Greater than | x > y |
| < | Less than | x < y |
| >= | Greater than or equal to | x >= y |
| <= | Less than or equal to | x <= y |

## Logical Operators

|  |  |  |
| --- | --- | --- |
| Operator | Description | Example |
| and | Returns True if both statements are true | x < 5 and  x < 10 |
| or | Returns True if one of the statements is true | x < 5 or x < 4 |
| not | Reverse the result, returns False if the result is true | not(x < 5 and x < 10) |

## Identity Operators

|  |  |  |
| --- | --- | --- |
| Operator | Description | Example |
| is | Returns true if both variables are the same object | x is y |
| is not | Returns true if both variables are not the same object | x is not y |

## Membership Operators

|  |  |  |
| --- | --- | --- |
| Operator | Description | Example |
| in | Returns True if a sequence with the specified value is present in the object | x in y |
| not in | Returns True if a sequence with the specified value is not present in the object | x not in y |

## Bitwise Operators

|  |  |  |
| --- | --- | --- |
| Operator | Name | Description |
| & | AND | Sets each bit to 1 if both bits are 1 |
| | | OR | Sets each bit to 1 if one of two bits is 1 |
| ^ | XOR | Sets each bit to 1 if only one of two bits is 1 |
| ~ | NOT | Inverts all the bits |
| << | Zero fill left shift | Shift left by pushing zeros in from the right and let the leftmost bits fall off |
| >> | Signed right shift | Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off |

## Escape Characters [[11]](#endnote-11)

|  |  |
| --- | --- |
| Character | Name |
| \’ | Single Quote |
| \\ | Backslash |
| \n | New Line |
| \r | Carriage Return |
| \t | Tab |
| \b | Backspace |
| \f | Form Feed |
| \ooo | Octal Value |
| \xhh | Hex Value |

## RegEx Library[[12]](#endnote-12)

RegEx is a Python built-in package that is used to check whether a string object contains a specified search pattern. To use RegEx, you must import the re module through the statement import re

* RegEx is short for “Regular Expression” which is a sequence of characters that forms a search pattern.

### RegEx Functions

|  |  |
| --- | --- |
| Function | Description |
| findall(regex, str) | Returns a list containing all matches **regex** in **str** |
| search(regex, str) | Returns a Match object if there is a match **regex** anywhere in the string **str** |
| split(regex, str) | Returns a list where the string **str** has been split at each match **regex** |
| sub(regex1, regex2, text) | Replaces one or many matches **regex1** in **text** with a string **regex2** |

### RegEx Metacharacters

|  |  |  |
| --- | --- | --- |
| Characters | Description | Example |
| [] | A set of characters | “[a-c]” |
| \ | Signals a special sequence | “\Z” |
| . | Any character (except newline character) | “be..n” |
| ^ | Starts with | “^be” |
| $ | Ends with | “ing$” |
| \* | Zero or more occurrences | “be.\*n” |
| + | One or more occurrences | “be.+n” |
| ? | Zero or one occurrences | “be.?n” |
| {} | Exactly the specified number of occurrences | “be.{1}n” |
| | | Either or | “yes|no” |
| () | Capture and group |  |

### RegEx Special Sequences

|  |  |  |
| --- | --- | --- |
| Character | Description | Example |
| \A | Returns a match if the specified characters are at the beginning of the string | “\ABehold” |
| \b | Returns a match where the specified characters are at the beginning or at the end of a word | “\bing”  “ing\b” |
| \B | Returns a match where the specified characters are present, but NOT at the beginning (or at the end) of a word | “\Bing”  “ing\B” |
| \d | Returns a match where the string contains digits | “\d” |
| \D | Returns a match where the string does not contain digits | “\D” |
| \s | Returns a match where the string contains a white space character | “\s” |
| \S | Returns a match where the string does not contain a white space character | “\S” |
| \w | Returns a match where the string contains any word characters | “\w” |
| \W | Returns a match where the string does not contain any word characters | “\W” |
| \Z | Returns a match if the specified characters are at the end of the string | “Bye\Z” |

# References

1. <https://www.tutorialspoint.com/python/python_basic_operators.htm> [↑](#endnote-ref-1)
2. <https://www.tutorialspoint.com/python/> [↑](#endnote-ref-2)
3. <https://www.geeksforgeeks.org/python-programming-language/> [↑](#endnote-ref-3)
4. <https://docs.python.org/3.5/library/stdtypes.html> [↑](#endnote-ref-4)
5. <https://medium.com/@meghamohan/mutable-and-immutable-side-of-python-c2145cf72747> [↑](#endnote-ref-5)
6. <https://www.geeksforgeeks.org/difference-method-function-python/> [↑](#endnote-ref-6)
7. <https://www.w3schools.com/python/python_lambda.asp> [↑](#endnote-ref-7)
8. <https://www.w3schools.com/python/python_iterators.asp> [↑](#endnote-ref-8)
9. <https://www.w3schools.com/python/python_lists_methods.asp> [↑](#endnote-ref-9)
10. <https://www.w3schools.com/python/python_operators.asp> [↑](#endnote-ref-10)
11. <https://www.w3schools.com/python/gloss_python_escape_characters.asp> [↑](#endnote-ref-11)
12. <https://www.w3schools.com/python/python_regex.asp> [↑](#endnote-ref-12)