Python

# BASICS

## 

Python about

Identifier names

Comment

Print

Input

Statement

Key word

## 

## 

## 

## 

## 

## 

## Values and Variables

### Value

* A value is the representation of some entity that can be manipulated by a program.Values are the most basic thing any program works with. (Ex. ‘Hello, World!’ or 1)
* Values belong to different data types.

### Data Types

These Data Types are;

| **Data Types** | | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Numeric** | **String** | **Sequence** | **Binary** | **Mapping** | **Boolean** | **Set** | **None** |
| integer | String | list | bytes | dictionary | boolean | set | none |
| float |  | tuple | bytearray |  |  | *frozenset* |  |
| complex |  | range | memoryview |  |  |  |  |

You can get the data type of any object by using the type ([value in question ]) function.

type(“hello world”)

### Variable

* Variable is a label for a location in memory. It can be used to hold any type of value (ex. an integer, string. list etc).
* A global variable is a variable that is defined in the main program and is accessible

throughout the file. Because of their wide-ranging effects should almost never use them. Only objects which are intended to be used globally, like functions and classes, should be put in the global namespace.

* A local variable is a variable defined inside a function and is only accessible to that function and exists as long as the function is executing. Most variables should be in functions.
* To initialise a variable in Python, we simply assign a value to a label (identifier name).
* We create a variable using the equal key( =)

year = 2023

Here we are assigning the integer value of 2023 to the memory location which is labelled as year.

## Data Types

### Numeric

Python numeric data type is used to hold numeric values. Python supports 3 types of numbers: integers, float and complex.

#### Integer (int)

* Int, or integer, is a whole number, positive or negative, without decimals, of unlimited length. (Ex. 1, 5, 1350 or -34)Integers are faster than floats in the cpu.

x = 1

Convert a value to a integer using **int()**

int([value])

#### Float (float)

Float, or "floating point number" is a number, positive or negative, containing one or more decimals. (Ex. 1.0, 5.32, 13.5 or -3.4)

x=1.0

Convert a value to a float using **float()**

float([value])

#### Complex(complex)

Complex numbers are written with a "j" as the imaginary part. (Ex. 5j, -5j 3+5j)

x=1j

Convert a value to a complex using **complex()** You cannot convert complex numbers into another number type.

complex([value])

Numeric Operators (integers and floats)

| Operation | Symbol | Example | Result |
| --- | --- | --- | --- |
| Addition | + | 28 + 10 | 38 |
| Subtraction | - | 28 - 10 | 18 |
| Multiplication | \* | 28 \* 10 | 280 |
| Float Division | / | 28/10 | 2.8 |
| Integer Division | // | 28 // 10 | 2 |
| Modulus (remainder) | % | 28 % 10 | 8 |
| Exponent (power) | \*\* | 28\*\*10 | 296196766695424 |

Order of operations BEDMAS (**B**rackets **E**xponents **D**ivision **M**ultiplication **A**ddition **S**ubtraction)

### Strings

#### Strings

A string is a sequence of characters or character. Strings in python are surrounded by either single quotation marks, or double quotation marks. (Ex. “Hello World”, ‘bye’, “D”).

Strings are immutable, meaning that they cannot change. We can only create a new string.

#### Create a string

name = ‘devo’

#### Get the number of elements in the string using len

The **len()** function returns the length of a string:

a = "Hello, World!"

print(len(a))

>13

#### Looping Through a String

Since strings are arrays, we can loop through the characters in a string, with a for loop.

for x in "yo":

print(x)

>y

>o

Convert a value to string using str()

### Boolean

* Boolean variables can have only two values - **True** or **False**.
* To make a boolean, you can assign a True or False value or create an expression that evaluates to True or False.

A = True

B = False

C = (1==3)

* We can evaluate values & variables and convert to the boolean data type using **bool()**

Almost any value is evaluated to True if it has some sort of content.

* Any string with content is True, But empty strings are False
* Any number is True, But the number 0 is False
* Any list, tuple, set, and dictionary are True, except empty ones.

#### Comparison/ Relational Operators

* Used for comparing values. It either returns True or False according to if the condition is correct or wrong. These operators are:

**a == b** a is equal to b

**a != b** is not equal to b

**a > b** a is greater than b

**a < b** a is less than b

**a >= b** a is greater than or equal to b

**a <= b** a is less than or equal to b

**a is b** a is the same as b this checks if they are the exact same object. This is a test of identity.

**a is not b** is not the same as b his checks if they are the exact same object. This is a test of identity.

#### Boolean/Logical Operators

* Boolean values can be manipulated by the use of boolean operators which include **and, or,** and **not**.

and - Evaluates to True when both arguments are true.

or - Evaluates to True when any of the arguments are true.

not - Only requires one argument and returns the opposite of the argument. If True then it equates to False. If False it equates to True

### Lists []

* List is a collection which is ordered and changeable. Allows duplicate members.
* List is a sequence of values. they can be of any data type.
* The values in list are called elements or sometimes items.

#### Creating A list

* The simplest way is to enclose the elements in square brackets (“[" and “]”) and elements are separated with a comma..
* You can assign list values to variables:

list1 = ["abc", 34, True, 40, "male"]

cheeses = ['Cheddar', 'Edam', 'Gouda']

#### List Length

* To determine how many items a list has, use the **len()** function

#### Traversing a list

for i in range(len(numbers)):

numbers[i] = numbers[i] \* 2

#### List Methods

| **Method** | **Description** |
| --- | --- |
| append() | Adds an element at the end of the list |
| clear() | Removes all the elements from the list |
| copy() | Returns a copy of the list |
| count() | Returns the number of elements with the specified value |
| extend() | Add the elements of a list (or any iterable), to the end of the current list |
| index() | Returns the index of the first element with the specified value |
| insert() | Adds an element at the specified position |
| pop() | Removes the element at the specified position |
| remove() | Removes the item with the specified value |
| reverse() | Reverses the order of the list |
| sort() | Sorts the list |

### Tuples ()

* Tuple is a collection which is ordered and unchangeable. Allows duplicate members.
* A tuple is a sequence of values much like a list. The values stored in a tuple can be any type, and they are indexed by integers. The important difference is that tuples are immutable. Tuples are also comparable and hashable so we can sort lists of them and use tuples as key values in Python dictionaries

#### Creating a Tuple:

* Tuples are made with round brackets ()
* thistuple = ("apple", "banana", "cherry")
* tupleexample2 = ("abc", 34, True, 40, "male")
* To create a tuple with a single element, you have to include the final comma:

Without the comma Python treats ('a') as an expression with a string in parentheses that evaluates to a string

#### Modifying Tuples

* You can’t modify such as adding, updating or deleting elements of a tuple. This is because tuples are immutable. But you can convert it to a list and then change it back to a tuple.

#### Loop through tuple

* You can loop through the tuple items by using a for loop.

thistuple = ("apple", "banana", "cherry")

for x in thistuple:

print(x)

#### List operators and methods

* Most list operators also work on tuples such as slicing and the len() function to see how many elements are in the tuple.

#### Tuple Methods

* Python has two built-in methods that you can use on tuples.

| **Method** | **Description** |
| --- | --- |
| count() | Returns the number of times a specified value occurs in a tuple |
| index() | Searches the tuple for a specified value and returns the position of where it was found |

### Set {}

* Set is a collection which is unordered, unchangeable, and unindexed. No duplicate members.

#### Creating Python Sets

* A set is created by placing all the items (elements) inside curly braces {}, separated by comma, or by using the built-in set() function.
* It can have any number of items and they may be of different types (integer, float, tuple, string etc.).
* But a set cannot have mutable elements like lists, sets or dictionaries as its elements.my\_set = {1, 2, 3}
* Empty curly braces {} will make an empty dictionary in Python. To make a set without any elements, we use the set() function without any argument.

a = set()

* We cannot access or change an element of a set using indexing or slicing. since sets are unordered, indexing has no meaning. len() and for loops still work

| **Method** | **Description** |
| --- | --- |
| add() | Adds an element to the set |
| clear() | Removes all the elements from the set |
| copy() | Returns a copy of the set |
| difference() | Returns a set containing the difference between two or more sets |
| difference\_update() | Removes the items in this set that are also included in another, specified set |
| discard() | Remove the specified item |
| intersection() | Returns a set, that is the intersection of two other sets |
| intersection\_update() | Removes the items in this set that are not present in other, specified set(s) |
| isdisjoint() | Returns whether two sets have a intersection or not |
| pop() | Removes an element from the set |
| remove() | Removes the specified element |
| symmetric\_difference() | Returns a set with the symmetric differences of two sets |
| symmetric\_difference\_update() | inserts the symmetric differences from this set and another |
| union() | Return a set containing the union of sets |
| update() | Update the set with the union of this set and others |

### Frozenset

* Frozen set is similar to set except in mutability. Python set is mutable itself, however, the elements in the set are immutable. In the frozen set, along with elements frozen set itself is immutable. Hence, We cannot add or remove items from frozen sets.
* Frozen sets can be used as keys in the dictionary as they are immutable and hashable.
* Frozen sets are created using built-in function frozenset( )

>>> py\_frozenset = frozenset([1,2,3,4,5])

### Dictionaries {}

* A dictionary is a collection which is ordered, changeable and do not allow duplicates.
* A dictionary is like a list, but more general. In a list, the index positions have to be integers; in a dictionary, the indices can be (almost) any type.
* You can think of a dictionary as a mapping between a set of indices (which are called keys) and a set of values. Each key maps to a value. The association of a key and a value is called a key-value pair or sometimes an item.
* As an example, we’ll build a dictionary that maps from English to Spanish words,so the keys and the values are all strings.

#### Creating a Dictionary

* Python dictionary can be created either by directly putting up comma-separated key-value pairs inside curly braces ‘{ }’

you can create a new dictionary with three items.

>>> eng2sp = {'one': 'uno', 'two': 'dos', 'three': 'tres'}

#### Dictionary Methods

Method Description

clear() Removes all the elements from the dictionary

copy() Returns a copy of the dictionary

fromkeys() Returns a dictionary with the specified keys and value

get() Returns the value of the specified key

items() Returns a list containing a tuple for each key value pair

keys() Returns a list containing the dictionary's keys

pop() Removes the element with the specified key

popitem() Removes the last inserted key-value pair

setdefault() Returns the value of the specified key.

update() Updates the dictionary with the specified key-value pairs

values() Returns a list of all the values in the dictionary

## 

Range()

Python **range()** is an in-built function in Python which returns a sequence of numbers starting from 0 and increments to 1 until it reaches a specified number.

We use **range()** function with for and while loop to generate a sequence of numbers. Following is the syntax of the function:

range(start, stop, step)

Here is the description of the parameters used:

* **start**: Integer number to specify starting position, (Its optional, Default: 0)
* **stop**: Integer number to specify starting position (It's mandatory)
* **step**: Integer number to specify increment, (Its optional, Default: 1)

range(6) is not the values of 0 to 6, but the values 0 to 5.

range(2,6) means values from 2 to 6 (but not including 6):

Range (2, 30, 3) means values from 2 to 6 (but not including 6) but it is incrmeted by 3.

## Selection/Conditional If statements

### If statements

* In order to write useful programs, we almost always need the ability to check conditions and change the behavior of the program accordingly. Conditional statements give us this ability.

The simplest form is the if statement:

if x > 0 :

print('x is positive')

* If the logical condition is true, then the indented statement gets executed. If the logical condition is false, the indented statement is skipped.

### Alternative if

* A second form of the if statement is alternative execution, in which there are two possibilities and the condition determines which one gets executed. The alternatives are called branches, because they are branches in the flow of execution.

The syntax looks like this:

if x%2 == 0 :

print('x is even')

else :

print('x is odd')

### Chained if conditionals

* Sometimes there are more than two possibilities and we need more than two branches. One way to express a computation like that is a chained conditional using elif. elif is an abbreviation of “else if.”.
* There is no limit on the number of elif statements. If there is an else clause, it has to be at the end, but there doesn’t have to be one.
* Each condition is checked in order. If the first is false, the next is checked, and so on. If one of them is true, the corresponding branch executes, and the statement ends. Even if more than one condition is true, only the first true branch executes:

if x < y:

print('x is less than y')

elif x > y:

print('x is greater than y')

elif:

print('x and y are equal')

## Iteration while and for loops

* The control flow of repeating statement(s) as iteration. Loops are the implementation of iteration within a programming language (It is called loops because it looks like a loop in a flowchart.). In Python there are two kinds of loops, while and for.

### While Loops

* While loops repadolaty execute statement(s) while its boolean condition is true and ends when its boolean condition is false.
* While loops are often referred to as being an event-controlled loop. This is because the while loop stops executing when an event( the boolean condition becoming false) occurs.

n = 5

while n > 0:

print(n)n = n - 1

print('Blastoff!')

#### Infinite loops

* Sometimes you don’t know it’s time to end a loop until you get half way through the body. In that case you can write an infinite loop on purpose and then use the break statement to jump out of the loop. This loop is obviously an infinite loop because the logical expression on the while statement is simply the logical constant True:

while True:

print(n, end=' ')n = n - 1

print('Done!')

### For Loops

* A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).
* When we have a list of things to loop through, we can construct a for loop using a for statement.

friends = ['Joseph', 'Glenn', 'Sally']

for friend in friends:

print('Happy New Year:', friend)

print('Done!')

### Functions

A function is a reusable block of programming statements designed to perform a certain task.

To define a function, use the def keyword.

### OOP

Object-Oriented Programming (OOP) breaks down a programming task into objects that expose data (attributes) and behaviour (methods/functions) using interfaces.

There are 4 Principles of OOP

1. Inheritance
2. Polymorphism
3. Encapsulation
4. Abstraction

An object is any entity that has attributes and behaviours. For example, a dog is an object. It has attributes (data) and behaviours (functions).

attributes - name, age, breed, etc.

behaviour - barking, tail-wagging, etc.

A class is a blueprint for an object. It defines the attributes and behaviours of what the object will contain.

#### Classes

To create a class we use the ‘class’ keyword. The class identifier name should start with a capital.

class Dog:

#Class Attribute

alias = “Canine”

def \_\_init\_\_(self,name, age, breed)

#Instance Attributes

self.name = name

self.age = age

self.breed = breed

#instance method

def bark():

#Instance

dog1 = Dog(“Champ”, 3, “Terrier”)

The argument always most have self.

Methods(Functions inside Classes)

OOP allows us to create our own data types. Everything in python is an object

Classes - Represent something such as a car.

Objects- Specific examples of a class. Such as Davonte’s mazda 3.

Classes have a set of attributes and methods

For example, a car class might contain something like its make, model, year and VIN. In addition, it will have behaviour/functionality like start, drive, park, reverse, off.

class Car:

def \_\_init\_\_(self, make, model, year, vin):

self.make = make

self.model = model

self.year = year

self.vin = vin

To create a class, use the keyword ‘class’ followed by the name of the class you want it to be called. It must start with a capital

Class Car:

The \_\_init\_\_ special method, also known as a Constructor, is used to initialize a class with attributes such as make, model, year, vin.

This class can be instantiated to any number of objects.

davontesCar = Car("Mazda","3", 2009, "sdnf43f94a")

momCar =Car("BMW","X5", 2015, "gensd24bjda")

The self parameter is a reference to the specific instance of the class, and is used to access variables that belongs to the class.