**Python**

* Python is an interpreted, high-level and general-purpose, dynamically typed programming language
* It is also Object oriented, modular oriented and a scripting language.
* In Python, everything is considered as an Object.
* A python file has an extension of .py
* Python follows Indentation to separate code blocks instead of flower brackets({}).
* We can run a python file by the following command in cmd(Windows) or shell(mac/linux).

$ python <filename.py> or $ python3 <filename.py>

**By default, python doesn't require any imports to run a python file.**

**Create and execute a program**

1. Open up a terminal/cmd
2. Create the program: nano/cat > nameProgram.py
3. Write the program and save it
4. python nameProgram.py

**Basic Datatypes**

| **Data Type** | **Description** |
| --- | --- |
| int | Integer values [0, 1, -2, 3] |
| float | Floating point values [0.1, 4.532, -5.092] |
| char | Characters [a, b, @, !, `] |
| str | Strings [abc, AbC, A@B, sd!, `asa] |
| bool | Boolean Values [True, False] |
| complex | Complex numbers [2+3j, 4-1j] |

**Keywords**

* As of python3.8 there are 35 keywords

| **Keyword** | **Description** | **Category** |
| --- | --- | --- |
| True | Boolean value for not False or 1 | Value Keyword |
| False | Boolean Value for not True or 0 | Value Keyword |
| None | No Value | Value keyword |
| and | returns true if both (oprand) are true (other language && ) | Operator keyword |
| or | returns true of either operands is true (other language |  |
| in | returns true if word is in iterator | Operator keyword |
| is | returns true if id of variables are same | Operator keyword |
| not | returns opposite Boolean value | Operator Keyword |
| if | get into block if expression is true | conditional |
| elif | for more than 1 if checks | conditional |
| else | this block will be executed if condition is false | conditional |
| for | used for looping | iteration |
| while | used for looping | iteration |
| break | get out of loop | iteration |
| continue | skip for specific condition | iteration |
| def | make user defined function | structure |
| class | make user defined classes | structure |
| lambda | make anonymous function | structure |
| with | execute code within context manager's scope | structure |
| as | alias for something | structure |
| pass | used for making empty structures(declaration) | structure |
| return | get value(s) from function, get out of function | returning keyword |
| yield | yields values instead of returning (are called generators) | returning keyword |
| import | import libraries/modules/packages | import |
| from | import specific function/classes from modules/packages | import |
| try | this block will be tried to get executed | exception handling |
| except | is any exception/error has occured it'll be executed | exception handling |
| finally | It'll be executed no matter exception occurs or not | exception handling |
| raise | throws any specific error/exception | exception handling |
| assert | throws an AssertionError if condition is false | exception handling |
| async | used to define asynchronous functions/co-routines | asynchronous programming |
| await | used to specify a point when control is taken back | asynchronous programming |
| del | deletes/unsets any user defined data | variable handling |
| global | used to access variables defined outside of function | variable handling |
| nonlocal | modify variables from different scopes | variable handling |

**Operators**

| **Operator** | **Description** |
| --- | --- |
| ( ) | grouping parenthesis, function call, tuple declaration |
| [ ] | array indexing, also declaring lists etc. |
| ! | relational not, complement, ! a yields true or false |
| ~ | bitwise not, ones complement, ~a |
| - | unary minus, - a |
| + | unary plus, + a |
| \* | multiply, a \* b |
| / | divide, a / b |
| % | modulo, a % b |
| + | add, a + b |
| - | subtract, a - b |
| << | shift left, left operand is shifted left by right operand bits (multiply by 2) |
| >> | shift right, left operand is shifted right by right operand bits (divide by 2) |
| < | less than, result is true or false, a %lt; b |
| <= | less than or equal, result is true or false, a <= b |
| > | greater than, result is true or false, a > b |
| >= | greater than or equal, result is true or false, a >= b |
| == | equal, result is true or false, a == b |
| != | not equal, result is true or false, a != b |
| & | bitwise and, a & b |
| ^ | bitwise exclusive or XOR, a ^ b |
| | | bitwise or, a |
| &&, and | relational and, result is true or false, a < b && c >= d |
| ||, or | relational or, result is true or false, a < b || c >= d |
| = | store or assignment |
| += | add and store |
| -= | subtract and store |
| \*= | multiply and store |
| /= | divide and store |
| %= | modulo and store |
| <<= | shift left and store |
| >>= | shift right and store |
| &= | bitwise and and store |
| ^= | bitwise exclusive or and store |
| |= | bitwise or and store |
| , | separator as in ( y=x,z=++x ) |

**Basic Data Structures**

**List**

* List is a collection which is ordered and changeable. Allows duplicate members.
* Lists are created using square brackets:

thislist = ["apple", "banana", "cherry"]

* List items are ordered, changeable, and allow duplicate values.
* List items are indexed, the first item has index [0], the second item has index [1] etc.
* The list is changeable, meaning that we can change, add, and remove items in a list after it has been created.
* To determine how many items a list has, use the len() function.
* A list can contain different data types:

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

* It is also possible to use the list() constructor when creating a new list

thislist = list(("apple", "banana", "cherry")) # note the double round-brackets

* pop() function removes the last value in the given list by default.
* thislist = ["apple", "banana", "cherry"]
* print(thislist.pop()) # cherry

print(thislist.pop(0)) #apple

**Tuple**

* Tuple is a collection which is ordered and unchangeable. Allows duplicate members.
* A tuple is a collection which is ordered and unchangeable.
* Tuples are written with round brackets.

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

* Tuple items are ordered, unchangeable, and allow duplicate values.
* Tuple items are indexed, the first item has index [0], the second item has index [1] etc.
* When we say that tuples are ordered, it means that the items have a defined order, and that order will not change.
* Tuples are unchangeable, meaning that we cannot change, add or remove items after the tuple has been created.
* Since tuple are indexed, tuples can have items with the same value:
* Tuples allow duplicate values:

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

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

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

print(len(thistuple))

* To create a tuple with only one item, you have to add a comma after the item, otherwise Python will not recognize it as a tuple.

thistuple = ("apple",)

print(type(thistuple))

# NOT a tuple

thistuple = ("apple")

print(type(thistuple))

* It is also possible to use the tuple() constructor to make a tuple.

thistuple = tuple(("apple", "banana", "cherry")) # note the double round-brackets

print(thistuple)

**Set**

* Set is a collection which is unordered and unindexed. No duplicate members.
* A set is a collection which is both unordered and unindexed.

thisset = {"apple", "banana", "cherry"}

* Set items are unordered, unchangeable, and do not allow duplicate values.
* Unordered means that the items in a set do not have a defined order.
* Set items can appear in a different order every time you use them, and cannot be referred to by index or key.
* Sets are unchangeable, meaning that we cannot change the items after the set has been created.
* Duplicate values will be ignored.
* To determine how many items a set has, use the len() method.

thisset = {"apple", "banana", "cherry"}

print(len(thisset))

* Set items can be of any data type:

set1 = {"apple", "banana", "cherry"}

set2 = {1, 5, 7, 9, 3}

set3 = {True, False, False}

set4 = {"abc", 34, True, 40, "male"}

* It is also possible to use the set() constructor to make a set.

thisset = set(("apple", "banana", "cherry")) # note the double round-brackets

* frozenset() is just an immutable version of Set. While elements of a set can be modified at any time, elements of the frozen set remain the same after creation.

set1 = {"apple", "banana", "cherry"}

frzset=frozenset(set1)

print(frzset)

**Dictionary**

* Dictionary is a collection which is unordered and changeable. No duplicate members.
* Dictionaries are used to store data values in key:value pairs.
* Dictionaries are written with curly brackets, and have keys and values:

thisdict = {

"brand": "Ford",

"model": "Mustang",

"year": 1964

}

* Dictionary items are presented in key:value pairs, and can be referred to by using the key name.

thisdict = {

"brand": "Ford",

"model": "Mustang",

"year": 1964

}

print(thisdict["brand"])

* Dictionaries are changeable, meaning that we can change, add or remove items after the dictionary has been created.
* Dictionaries cannot have two items with the same key.
* Duplicate values will overwrite existing values.
* To determine how many items a dictionary has, use the len() function.

print(len(thisdict))

* The values in dictionary items can be of any data type

thisdict = {

"brand": "Ford",

"electric": False,

"year": 1964,

"colors": ["red", "white", "blue"]

}

* pop() Function is used to remove a specific value from a dictionary. You can only use key bot the value. Unlike Lists you have to give a value to this function
* car = {
* "brand": "Ford",
* "model": "Mustang",
* "year": 1964
* }
* x = car.pop("model")
* print(x)# Mustang

print(car)#{'brand': 'Ford', 'year': 1964}

**Conditional branching**

if condition:

pass

elif condition2:

pass

else:

pass

**Loops**

Python has two primitive loop commands:

1. while loops
2. for loops

**While loop**

* With the while loop we can execute a set of statements as long as a condition is true.
* Example: Print i as long as i is less than 6

i = 1

while i < 6:

print(i)

i += 1

* The while loop requires relevant variables to be ready, in this example we need to define an indexing variable, i, which we set to 1.
* With the break statement we can stop the loop even if the while condition is true
* With the continue statement we can stop the current iteration, and continue with the next.
* With the else statement we can run a block of code once when the condition no longer is true.

**For loop**

* A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).
* This is less like the for keyword in other programming languages, and works more like an iterator method as found in other object-orientated programming languages.
* With the for loop we can execute a set of statements, once for each item in a list, tuple, set etc.

fruits = ["apple", "banana", "cherry"]

for x in fruits:

print(x)

* The for loop does not require an indexing variable to set beforehand.
* To loop through a set of code a specified number of times, we can use the range() function.
* The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.
* The range() function defaults to increment the sequence by 1, however it is possible to specify the increment value by adding a third parameter: range(2, 30, 3).
* The else keyword in a for loop specifies a block of code to be executed when the loop is finished. A nested loop is a loop inside a loop.
* The "inner loop" will be executed one time for each iteration of the "outer loop":

adj = ["red", "big", "tasty"]

fruits = ["apple", "banana", "cherry"]

for x in adj:

for y in fruits:

print(x, y)

* for loops cannot be empty, but if you for some reason have a for loop with no content, put in the pass statement to avoid getting an error.

for x in [0, 1, 2]:

pass

**Function definition**

def function\_name():

return

**Function call**

function\_name()

* We need not to specify the return type of the function.
* Functions by default return None
* We can return any datatype.