**Python Learnings:**

a) a single line comment , #

b) multiline comment, “”” xxxx “””

**Variable Declaration:**

x = 5;

x=’shankar’; or x = “shankar” //both declarations are same

print(x)

y = 5.0;

print(y); // print is used to output variables

x = "awesome"  
print("Python is " + x)

**//Adding variables**

x = "Python is "  
y = "awesome"  
z =  x + y  
print(z)

**Variable Declarations:**

a) A variable name must start with a letter or underscore character

b) A variable name cannot start with a number

c) A variable name can only contain alpha-numeric characters and underscores

d) Variable names are case-sensitive (age, Age, and AGE are 3 different variables).

#Legal variable names:  
myvar = "John"  
my\_var = "John"  
\_my\_var = "John"  
myVar = "John"  
MYVAR = "John"  
myvar2 = "John"  
  
#Illegal variable names:  
2myvar = "John"  
my-var = "John"  
my var = "John"

**Assign Value to Multiple Variables:**

**x,y,z = “Orange”, “Banana”, “Apple”**

print(x)

print(y)

print(z)

Or

**x = y = z = "Orange"**

**Global Variables:**

a) Use key word **“global”** to make variable as global, global can be given to a variable which is declared inside a function also.

**Built in Data Types:**

Variables can store data of different types, and different types can do different things

Text Type: str

Numeric Type: int, float, **complex**

Sequence Type: list, tuple, range

Mapping Type: dict

Set Types: set, frozenset

Boolean Type: bool

Binary Types: bytes, bytearray, memoryview

x = 5;

print(type(x)) //output as **int**

Ex:

x = "Hello World" str

x = 20 int

x = 20.5 float

x = 1j complex

x = ["apple", "banana", "cherry"] list

x = ("apple", "banana", "cherry") tuple

x = range(6) range

x = {"name" : "John", "age" : 36} dict

x = {"apple", "banana", "cherry"} set

x = frozenset({"apple", "banana", "cherry"}) frozenset

x = True bool

x = b"Hello" bytes

x = bytearray(5) bytearray

x = memoryview(bytes(5)) memoryview

**Random Number:**

Python does not have a random() function to make a random number, but Python has a built-in module called random that can be used to make random numbers:

*import random  
print(random.randrange(1, 10))*

**String Literals**

String literals in python are surrounded by either single quotation marks, or double quotation marks.

'hello' is the same as "hello".

You can display a string literal with the print() function:

**Multiline Strings:**

You can assign a multiline string to a variable by using three quotes:

a = """Lorem ipsum dolor sit amet,  
consectetur adipiscing elit,  
sed do eiusmod tempor incididunt  
ut labore et dolore magna aliqua."""  
print(a)

three single quotes:

a = '''Lorem ipsum dolor sit amet,  
consectetur adipiscing elit,  
sed do eiusmod tempor incididunt  
ut labore et dolore magna aliqua.'''  
print(a)

**Note:** in the result, the line breaks are inserted at the same position as in the code.

**Boolean Values**

You can evaluate any expression in Python, and get one of two answers, True or False.

print(10 > 9)

print(10 == 9)

Or

a = 10

b = 20

if b>a:

print(“a is greaterthan b”)

else:

print(“b is greater than a”)

You can evaluate any expression in Python, and get one of two answers, True or False.

**Evaluate Values and Variables:**

The bool() function allows you to evaluate any value, and give you True or False in return,

x = "Hello"  
y = 15  
  
print(bool(x))  
print(bool(y))

## Most Values are True

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

Any string is True, except empty strings.

Any number is True, except 0.

Any list, tuple, set, and dictionary are True, except empty ones.

## Some Values are False

In fact, there are not many values that evaluates to False, except empty values, such as (), [], {}, "", the number 0, and the value None. And of course the value False evaluates to False.

### **Example**

The following will return False:

bool(False)  
bool(None)  
bool(0)  
bool("")  
bool(())  
bool([])  
bool({})

**Type Casting:**

a) int() : constructs an integer from a float literal(by rounding down to previous whole number or a string literal (providing the string represents whole number) or from an integer literal.

ex: *x = int(1)*

*x = int(2.8)*

*x = int(“3”)*

b) strings:

*x = str("s1") # x will be 's1'  
y = str(2)    # y will be '2'  
z = str(3.0)  # z will be '3.0'*

c) float:

constructs a float number from an integer literal, a float literal or a string literal (providing the string represents a float or an integer)

x = float(1)     # x will be 1.0  
y = float(2.8)   # y will be 2.8  
z = float("3")   # z will be 3.0  
w = float("4.2") # w will be 4.2

**Strings:** single quotes or double quotes for a single line string. For a multiline string provide in “”” or ‘’’’

Like many other popular programming languages, strings in Python are arrays of bytes representing unicode characters.

However, Python does not have a character data type, a single character is simply a string with a length of 1.

Square brackets can be used to access elements of the string.

a = "Hello, World!"  
print(a[1])

**Slicing:** b = “Hello World”

print(b[2:5])

**Negative Indexing:** Use negative indexes to start the slice from the end of the string:

b = "Hello, World!"  
print(b[-5:-2])

**String Length:** Use len() function to get the length of the string

a = "Hello, World!"  
print(len(a))

**String Methods:**

a) string() method: Used to remove any whitespace from the beginning or end.

a = “ Hello, world! “

print(a.string()) # returns “Hello, World”

b) lower(), converts the string to lower case.

c) upper()

d) replace : a = “hello, world”, print(a.replace(“H”, “J”))

e) Check String:

txt = "The rain in Spain stays mainly in the plain"  
x = "ain" in txt  
print(x)

f) The strip() method removes any whitespace from the beginning or the end:

a = " Hello, World! "  
print(a.strip()) # returns "Hello, World!"

g) To check if a certain phrase or character is present in a string, we can use the keywords in or not in. (find equivalent in c++)

h) Concatenation Strings:

a = “hello”

b = “world”

c = a + b

print(c), adding a space : d = a + “ “ + b

Note: To append integers to the string , use format method, The format() method takes unlimited number of arguments, and are placed into the respective placeholders:

*age = 36  
txt = "My name is John, and I am {}"  
print(txt.format(age))*

*quantity = 3  
itemno = 567  
price = 49.95  
myorder = "I want {} pieces of item {} for {} dollars."  
print(myorder.format(quantity, itemno, price))*

**Boolean in Python:**

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

Any string is True, except empty strings.

Any number is True, except 0.

Any list, tuple, set, and dictionary are True, except empty ones.

In fact, there are not many values that evaluates to False, except empty values, such as (), [], {}, "", the number 0, and the value None. And of course the value False evaluates to False.

**Functions can return bool:**

*def myFunction() :  
  return True  
  
print(myFunction())*

bool(123) //True

bool([“shaknar”, “shankar”])

bool(()) //False

bool(“”) //False

bool(0) //False

**Class or Objects:**

*class Person:  
  def \_\_init\_\_(self, name, age):  
    self.name = name  
    self.age = age  
  
  def myfunc(self):  
    print("Hello my name is " + self.name)  
  
p1 = Person("John", 36)*

*p1.age = 50  
p1.myfunc()*

*delete p1.age //Deleting the age property of the class*

*del p1 //Delete the object*

**Note:** The self parameter is a reference to the current instance of the class, and is used to access variables that belong to the class.

It does not have to be named self , you can call it whatever you like, but it has to be the first parameter of any function in the class.

*class Person:  
  def \_\_init\_\_(mysillyobject, name, age):  
    mysillyobject.name = name  
    mysillyobject.age = age  
  
  def myfunc(abc):  
    print("Hello my name is " + abc.name)  
  
p1 = Person("John", 36)  
p1.myfunc()*

**Pass Statement:**  class definitions cannot be empty, butif you for some reason if you have a class definition with no content, put the **pass** statement to avoid getting an error.

*class Person:*

*pass*

**Python Inheritance:**  Inheritance allows us to define a class that inherits all methods and properties from another class.

*class Person:*

*def \_\_\_init\_\_(self, fname, lname):*

*self.firstname = fname*

*self.lastname = lname*

*def printname(self):*

*print(self.firstname, self.lastname)*

*x = Person(“John”, “Doe”)*

*x.printname*

**Child Class Creation:**

*class Student(Person):*

*pass*

*x = Student(“Mike”, “Olsen”)*

*x.printname()*

**Add the \_\_init\_\_() function:** (When the we add the \_\_init\_\_ function to the child class, child class will no longer inherit the parent class \_\_init\_\_ function. It means child class init overrides the parent’s init function.

To keep the inheritance of the parent’s \_\_init\_\_(), add a call in child class as below:

*class Student(Person):*

*def \_\_init\_\_(self, fname, lname):*

*Person.\_\_init\_\_(self,fname,lname)*

**Super***:* By using the super() function, you do not have to use the name of the parent element, it will automatically inherit the methods and properties from its parent.

## Add Properties or Methods to overridden function:

## Iterators:

## An iterator is an object which implements the iterator protocol, which consist of the methods \_\_iter\_\_() and \_\_next\_\_().

## Iterator vs Iterable

Lists, tuples, dictionaries, and sets are all iterable objects. They are iterable *containers* which you can get an iterator from.

All these objects have a iter() method which is used to get an iterator:

*mytuple = ("apple", "banana", "cherry")  
myit = iter(mytuple)  
  
print(next(myit))  
print(next(myit))  
print(next(myit))*

Strings are also Iterable Objects, containing a sequence of characters:

**Looping through iterator:**

mytuple = ("apple", "banana", "cherry")  
  
for x in mytuple:  
  print(x)

## Create an Iterator:

To create an object/class as an iterator you have to implement the methods \_\_iter\_\_() and \_\_next\_\_() to your object.

As you have learned in the [Python Classes/Objects](https://www.w3schools.com/python/python_classes.asp) chapter, all classes have a function called \_\_init\_\_(), which allows you to do some initializing when the object is being created.

The \_\_iter\_\_() method acts similar, you can do operations (initializing etc.), but must always return the iterator object itself.

The \_\_next\_\_() method also allows you to do operations, and must return the next item in the sequence.

## StopIteration:

To prevent the iteration to go on forever, we can use the StopIteration statement.

In the \_\_next\_\_() method, we can add a terminating condition to raise an error if the iteration is done a specified number of times:

*/\*\*\*\**

*def \_\_iter\_\_(self):  
    self.a = 1  
    return self  
  
  def \_\_next\_\_(self):  
    if self.a <= 20:  
      x = self.a  
      self.a += 1  
      return x  
    else:  
      raise StopIteration  
  
myclass = MyNumbers()  
myiter = iter(myclass)  
  
for x in myiter:  
  print(x)*

*\*\*\*/*

**Mutliple Else condition in Python:**

**if a<b:**

**elif a <20:**

**else:**

**Module:**

Consider a module to be the same as a code library.

A file containing a set of functions you want to include in your application.

To create a module just save the code you want in a file with the file extension .py:

//mymodule.py

*def greeting(name):  
  print("Hello, " + name)*

**Use a Module (use import statement):**

**Note:** When using a function from a module, use the syntax: *module\_name.function\_name*.

A module can contain variables of all types(arrays, dictionaries, objects etc..)

A module file must contain an extension with .py.

**Build in Modules:**

import platform

**Import from Module:**

**File Handling:**

a) open() method to open a file, 4 modes, r—read(error will be thrown if the file does not exiting), w – write, a—append(create if the file does not exist), x—create mode.

2 other modes, to open the file, “t” – Text – Default Vaule – Text Mode

“b” – Binary – Binary Mode (e.g Images)

f = open(“data.txt”, “rt”),

f = open(“data.txt) ‘’’ by default in read mode the file will be opened’’’, here f is file Object

🡪 **read**() method to read the content of the file

print(f.read()) //by default it reads completes data file.

//Read only particular no. of characters

f = open(“data.txt”,”r”)

print(f.read(5)

🡪readline() //To read a line

print(f.readline()) //reads the 1st line

**Loop through line by line:**

for x in f:

print (x)

**Note:** Good Practice is to close the file Object. (f.close()),  You should always close your files, in some cases, due to buffering, changes made to a file may not show until you close the file.

write(): to write the data into the file.

**Remove a file:**

import os

os.remove(“demofile.txt”)

**To check if a file exists and then delete:**

import os

if os.path.exists(“demofile.txt”):

os.remove(“demofile.txt”)

else:

print(“the file does not exist”)

To delete a folder:

import os

os.rmdir(“my folder”)

**Note:** You can only remove *empty* folders. // Question: **How to remove non empty folders in Python?**

### Local and Global Variables in Functions

Variable names are by default local to the function, in which they get defined.

**def** f():

print(s)

s = "Python"

f()

Python

**def** f():

s = "Perl"

print(s)

f()

Perl

s = "Python"

f()

print(s)

Perl

Python

**Lamda:**

a) Lamda function is a small anonymous function.

b) A lambda function can take any number of arguments, but can only have one expression.

syntax: lambda *arguments*: *expression*

*x= lamda a: a+10*

*print(x(5)*

*x = lamda a,b,c: a+b+c*

The power of lambda is better shown when you use them as an anonymous function inside another function.

Say you have a function definition that takes one argument, and that argument will be multiplied with an unknown number:

def myfunc(n):  
  return lambda a : a \* n  
  
mydoubler = myfunc(2)  
mytripler = myfunc(3)  
  
print(mydoubler(11)) //22  
print(mytripler(11)) //33

**Arrays:**  Python does not have built-in support for Arrays, but python-Lists can be used instead.

**Note:** This page shows you how to use LISTS as ARRAYS, however, to work with arrays in Python you will have to import a library, like the [NumPy library](https://www.w3schools.com/python/numpy_intro.asp).

cars = [“shank”,”mad”,”kris”]

Access element is : print(cars[0])

Modify the element is: cars[0] = “shan1”

length of array is: len(cars)

pop //Used to remove the element based on the index, pop(1)

remove// Used to removed the element based on the match , remove(“shan1”)

**Note:** The list's remove() method only removes the first occurrence of the specified value.

Bytes to String conversion:

## '''bytes to string conversion'''

## y = b'shankar'

## a = str(y,'utf-8')

## print(a)

## Array Methods

Python has a set of built-in methods that you can use on lists/arrays.

|  |  |
| --- | --- |
| Method | Description |
| [append()](https://www.w3schools.com/python/ref_list_append.asp) | Adds an element at the end of the list |
| [clear()](https://www.w3schools.com/python/ref_list_clear.asp) | Removes all the elements from the list |
| [copy()](https://www.w3schools.com/python/ref_list_copy.asp) | Returns a copy of the list |
| [count()](https://www.w3schools.com/python/ref_list_count.asp) | Returns the number of elements with the specified value |
| [extend()](https://www.w3schools.com/python/ref_list_extend.asp) | Add the elements of a list (or any iterable), to the end of the current list |
| [index()](https://www.w3schools.com/python/ref_list_index.asp) | Returns the index of the first element with the specified value |
| [insert()](https://www.w3schools.com/python/ref_list_insert.asp) | Adds an element at the specified position |
| [pop()](https://www.w3schools.com/python/ref_list_pop.asp) | Removes the element at the specified position |
| [remove()](https://www.w3schools.com/python/ref_list_remove.asp) | Removes the first item with the specified value |
| [reverse()](https://www.w3schools.com/python/ref_list_reverse.asp) | Reverses the order of the list |
| [sort()](https://www.w3schools.com/python/ref_list_sort.asp) | Sorts the list |

**Note:** Python does not have built-in support for Arrays, but Python Lists can be used instead.

**List Vs Tuple in Python:**

[**List**](https://www.geeksforgeeks.org/python-list/) are just like the arrays, declared in other languages. Lists need not be homogeneous always which makes it a most powerful tool in Python. In Python, the list is a type of container in Data Structures, which is used to store multiple data at the same time. Lists are a useful tool for preserving a sequence of data and further iterating over it.  
**Syntax:**

list\_data = ['an', 'example', 'of', 'a', 'list']

[Tuple](https://www.geeksforgeeks.org/tuples-in-python/) is also a sequence data type that can contain elements of different data types, but these are immutable in nature. In other words, a tuple is a collection of Python objects separated by commas. The tuple is faster than the list because of static in nature.  
**Syntax:**

tuple\_data = ('this', 'is', 'an', 'example', 'of', 'tuple')

tuple\_data[0] = “shank” //gives the error.

|  |  |  |  |
| --- | --- | --- | --- |
|  | List | Tuple |  |
| 1 | Lists are mutable | Tuple are immutable |
| 2 | Implication of iterations is Time-consuming | Implication of iterations is comparatively Faster |
| 3 | The list is better for performing operations, such as insertion and deletion. | Tuple data type is appropriate for accessing the elements |
| 4 | Lists consume more memory | Tuple consume less memory as compared to the list |
| 5 | Lists have several built-in methods | Tuple does no have must built-in methods. |
| 6 | The unexpected changes and errors are more likely to occur | In tuple, it is hard to take place. |

### 8) Define PYTHON PATH?

**Ans:** PYTHONPATH is an environmental variable that is used when we import a module. Suppose at any time we import a module, PYTHONPATH is used to check the presence of the modules that are imported in different directories. Loading of the module will be determined by interpreters.

### 10) What do you understand by the term PEP 8?

**Ans:** PEP 8 is the Python latest coding convention and it is abbreviated as Python Enhancement Proposal. It is all about how to format your Python code for maximum readability.

### 11) How memory management is done in Python?

**Ans:**

* In Python memory management is done using private heap space. The private heap is the storage area for all the data structures and objects. The interpreter has access to the private heap and the programmer cannot access this private heap.
* The storage allocation for the data structures and objects in Python is done by the memory manager. The access for some tools is provided by core API for programmers to code.
* The built-in garbage collector in Python is used to recycle all the unused memory so that it can be available for heap storage area.

### 12) Java vs Python

**Ans:** The major difference between [Java](https://mindmajix.com/core-java) and Python are as follows:

|  |  |  |
| --- | --- | --- |
| **Function** | **Java** | **Python** |
| **Coding** | In Java, we need to write a long code to print something. | In Python coding is simple and smaller when compared to Java |
| **Syntax** | In Java we need to put a semicolon at the end of the statement and also code must be placed in curly braces. | Whereas, in Python indentation is mandatory as it improves the readability of the code. |
| **Dynamic** | In Java, we need to declare the type for each variable | **In this case, codes are dynamically typed and this is also known as “duck typing”** |
| **Easy to use** | Java is not easy to use because of its larger coding | In Python, it is very easy to code and perform very easily. |
| **Databases** | Java Database Connectivity (JDBC) is more popular and used most commonly. | In Python database access layers are weaker when compared to Java. |

Memory management in Python involves a private heap containing all Python objects and data structures. The management of this private heap is ensured internally by the *Python memory manager*. The Python memory manager has different components which deal with various dynamic storage management aspects, like sharing, segmentation, preallocation or caching.

### 14) What are the built-in types available in Python?

**Ans:** The built-in types in Python are as follows:

* Integer
* Complex numbers
* Floating-point numbers
* Strings
* Built-in functions

### 15) What **are Python Decorators**?

**Ans:** Decorator is the most useful tool in Python as it allows programmers to alter the changes in the behavior of class or function.

**Example for Python Decorator is:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1  2  3 | @gfg\_decorator  def hi\_decorator():      print("Gfg")  In Python, functions are the first class objects, which means that –   * Functions are objects; they can be referenced to, passed to a variable and returned from * other functions as well. * Functions can be defined inside another function and can also be passed as argument to another * function.   [Decorators](https://www.geeksforgeeks.org/function-decorators-in-python-set-1-introduction/) are very powerful and useful tool in Python since it allows programmers to modify the behavior of function or class. Decorators allow us to wrap another function in order to extend the behavior of wrapped function, without permanently modifying it.  In Decorators, functions are taken as the argument into another function and then called inside the wrapper function.  **Syntax for Decorator:**   |  | | --- | | @gfg\_decorator  def hello\_decorator():      print("Gfg")    '''Above code is equivalent to -    def hello\_decorator():      print("Gfg")    hello\_decorator = gfg\_decorator(hello\_decorator)''' |   In the above code, gfg\_decorator is a callable function, will add some code on the top of some another callable function, hello\_decorator function and return the wrapper function.  <https://www.geeksforgeeks.org/decorators-in-python/>. //Refer the challenge here. 16) How do we find bugs and statistical problems in Python? **Ans:**We can detect bugs in python source code using a static analysis tool named **PyChecker**. Moreover, there is another tool called **PyLint** that checks whether the Python modules meet their coding standards  or not. 17) What is the difference between .py and .pyc files? **Ans:**.py files are Python source files. .pyc files are the compiled bytecode files that are generated by the Python compiler 24) Define slicing in Python? **Ans:** Slicing is a procedure used to select a particular range of items from sequence types such as Strings, lists, and so on. 25) How can Python be an interpreted language? **Ans:** As in Python the code which we write is not machine-level code before runtime so, this is the reason why Python is called as an interpreted language. 24) Define **slicing** in Python? **Ans:** Slicing is a procedure used to select a particular range of items from sequence types such as Strings, lists, and so on.  The slice() function returns a slice object.  A slice object is used to specify how to slice a sequence. You can specify where to start the slicing, and where to end. You can also specify the step, which allows you to e.g. slice only every other item.  Syntax  slice(*start*, *end, step*)  Parameter Values   |  |  | | --- | --- | | **Parameter** | **Description** | | *start* | Optional. An integer number specifying at which position to start the slicing. Default is 0 | | *end* | An integer number specifying at which position to end the slicing | | *step* | Optional. An integer number specifying the step of the slicing. Default is 1 |   a = ("a", "b", "c", "d", "e", "f", "g", "h") x = slice(0, 8, 3) (start,end,step) print(a[x]) //output is: (‘a’,’d’,’g’) 25) How can Python be an interpreted language? **Ans:** As in Python the code which we write is not machine-level code before runtime so, this is the reason why Python is called as an interpreted language. 26) What happens when a function doesn’t have a return statement? Is this valid? **Ans:**Yes, this is valid. The function will then return a None object. The end of a function is defined by the block of code is executed (i.e., the indenting) not by any explicit keyword. 27) Define package in Python? **Ans:** In Python packages are defined as the collection of different modules.  **28) How can we make a Python script executable on Unix?**  **Ans:** In order to make a Python script executable on [Unix](https://mindmajix.com/unix-shell-scripting), we need to perform two things. They are:  Script file mode must be executable and  The first line should always begin with #.  **29) Which command is used to delete files in Python?**  **Ans:** OS.unlink(filename) or OS.remove(filename) are the commands used to delete files in Python Programming.  **Example:**   |  |  | | --- | --- | | 1  2 | import OS  OS.remove("abc.txt")  **30) Define pickling and unpickling in Python?**  **Ans:**  **Pickling in Python:** The process in which the pickle module accepts various Python objects and converts into a string representation and dumps the file accordingly using dump function is called pickling.  **Unpickling in Python:** The process of retrieving actual Python objects from the stored string representation is called unpickling.  **DefaultDict:** | |

**Lists:**

lst = ["Vienna", "London", "Paris", "Berlin", "Zurich", "Hamburg"]

print(lst[0])

print(lst[2])

print(lst[-1])

x = len(lst) //5

|  |  |
| --- | --- |
| **List** | **Description** |
| [] | An empty list |
| [1,1,2,3,5,8] | A list of integers |
| [42, "What's the question?", 3.1415] | A list of mixed data types |
| ["Stuttgart", "Freiburg", "München", "Nürnberg", "Würzburg", "Ulm","Friedrichshafen", Zürich", "Wien"] | A list of Strings |
| [["London","England", 7556900], ["Paris","France",2193031], ["Bern", "Switzerland", 123466]] | A nested list |
| ["High up", ["further down", ["and down", ["deep down", "the answer", 42]]]] | A deeply nested list |

a) a = input() //to read from the console

//Creating 2 dimensional array list

lt=[]

for a in range(4):

l = []

for b in range(3):

print("Enter the input:")

x = input()

l.append(x)

print("inner list is: ", l)

lt.append(l)

**Different b/w Extend and append:**

If you append a list to one more list, it wont be appended properly, If you do extend it will work

**pop() vs pop(i):** pop() removes the last element , pop(i) removes the particular element with index

**remove: Remove an element with value . ex: list.remove(“shan”)**

### Find the Position of an Element in a List

The method "index" can be used to find the position of an element within a list:

s.index(x[, i[, j]])

It returns the first index of the value x. A ValueError will be raised, if the value is not present. If the optional parameter i is given, the search will start at the index i. If j is also given, the search will stop at position j.

colours = ["red", "green", "blue", "green", "yellow"]

colours.index("green")

Output::

1

colours.index("green", 2)

Output::

3

colours.index("green", 3,4)

Output::

3

colours.index("black")

Try and except in python:

try:  
  f = open("demofile.txt")  
  f.write("Lorum Ipsum")  
except:  
  print("Something went wrong when writing to the file")  
finally:  
  f.close()

## Raise an exception:

### **Example**

Raise an error and stop the program if x is lower than 0:

x = -1  
  
if x < 0:  
  raise Exception("Sorry, no numbers below zero")

The **id() function** (print the memory address) shows us that both variables point at the same list object, i.e. they share this object.

This list is a so-called "shallow list", because it doesn't have a nested structure, i.e. no sublists are contained in the list. In the next step we assign colours1 to colours2.

Below is the tabular Difference between the Shallow Copy and Deep Copy:

| **SHALLOW COPY** | **DEEP COPY** |
| --- | --- |
| Shallow Copy stores the references of objects to the original memory address. | Deep copy stores copies of the object’s value. |
| Shallow Copy reflects changes made to the new/copied object in the original object. | Deep copy doesn’t reflect changes made to the new/copied object in the original object. |
| Shallow Copy stores the copy of the original object and points the references to the objects. | Deep copy stores the copy of the original object and recursively copies the objects as well. |
| Shallow copy is faster. | Deep copy is comparatively slower. |

**Dictionaries:**

what's the difference between lists and dictionaries? A list is an ordered sequence of objects, whereas dictionaries are unordered sets. However, the main difference is that items in dictionaries are accessed via keys and not via their position.

Dictionaries are the Python implementation of an abstract data type, known in computer science as an associative array. Associative arrays consist - like dictionaries of (key, value) pairs, such that each possible key appears at most once in the collection. Any key of the dictionary is associated (or mapped) to a value. The values of a dictionary can be any type of Python data. So, dictionaries are unordered key-value-pairs. Dictionaries are implemented as hash tables, and that is the reason why they are known as "Hashes" in the programming language Perl.

a) if we try to access a key, i.e. a city, which is not contained in the dictionary? We raise a **KeyError:**

b) Keys of a dictionary are unique. In casse a keys is defined multiple times, the value of the last "wins":

food = {"bacon" : "yes", "spam" : "yes", "egg" : "yes", "spam" : "no" }

food

Output::

{'bacon': 'yes', 'spam': 'no', 'egg': 'yes'}

c) We can use arbitrary types as values in a dictionary, but there is a restriction for the keys. **Only immutable data types can be used as keys, i.e. no lists or dictionaries can be used: If you use a mutable data type as a key, you get an error message:**

**dic = { [1,2,3]: "abc"}**

---------------------------------------------------------------------------

TypeError Traceback (most recent call last)

<ipython-input-18-af2a40fe8efa> in <module>

----> 1 dic = { [1,2,3]:"abc"}

Tuple as keys are okay, as you can see in the following example:

dic = { (1,2,3):"abc", 3.1415:"abc"}

dic

Output::

| **Operator** | **Explanation** |
| --- | --- |
| len(d) | returns the number of stored entries, i.e. the number of (key,value) pairs. |
| del d[k] | deletes the key k together with his value |
| k in d | True, if a key k exists in the dictionary d |
| k not in d | True, if a key k doesn't exist in the dictionary d |

e) A dictionary can be copied with the method copy():

Shallow and Deep Copy concepts are applicable to the Dictionaries.

**Tuples:**

A tuple is a collection which is ordered and unchangeable. In Python tuples are written with round brackets.

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

**Access Tuple data:** print(thistuple[1])

**Negative indexing**: print(thistuple[-1])

**Access Range:** print(thistuple[2:5])

**Note**: Once a tuple is created, you cannot change its values. Tuples are unchangeable, or immutable as it also is called.

But there is a workaround. You can convert the tuple into a list, change the list, and convert the list back into a tuple.

**Check if Item Exists:**

**To determine if a specified item is present in a tuple use the in keyword:**

thistuple = ("apple", "banana", "cherry")  
if "apple" in thistuple:  
  print("Yes, 'apple' is in the fruits tuple")

**Create Tuple With One Item:**

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))

**NOTE:** You cannot remove items in a tuple. The **del** keyword can delete the tuple completely:

## Join Two Tuples:

tuple1 = ("a", "b" , "c")  
tuple2 = (1, 2, 3)  
  
tuple3 = tuple1 + tuple2  
print(tuple3)

## Sets:

A set is a collection which is unordered and unindexed. In Python, sets are written with curly brackets.

**Note:** Sets are unordered, so you cannot be sure in which order the items will appear.

## Change Items

Once a set is created, you cannot change its items, but you can add new items.

**Regular Expressions:**

Module : re (import re)

import re  
  
txt = "The rain in Spain"  
x = re.search("^The.\*Spain$", txt)

a) The **findall()** function returns a list containing all matches.

If no matches are found, an empty list is returned.

b) The search() function searches the string for a match, and returns a [Match object](https://www.w3schools.com/python/python_regex.asp#matchobject) if there is a match.

c) If there is more than one match, only the first occurrence of the match will be returned:

d) If no matches are found, the value None is returned:

e) The split() function returns a list where the string has been split at each match

f) Split the string only at the first occurrence:

*import re  
  
txt = "The rain in Spain"  
x = re.split("\s", txt, 1)  
print(x)*

g) The **sub()** function replaces the matches with the text of your choice.

h) [xyz] means e.g. either an "x", an "y" or a "z"

i) [a-z] means [abcdefghijklmnopqrstuvwxyz]

j) [-a-z] means – and [abcdefghijlklmnopqrstuvwyz]

h) You can control the number of replacements by specifying the count parameter:

*import re  
  
txt = "The rain in Spain"  
x = re.sub("\s", "9", txt, 2)  
print(x)*

j) **Predefined Character Classes**

You might have realized that it can be quite cumbersome to construe certain character classes. A good example is the character class, which describes a valid word character. These are all lower case and uppercase characters plus all the digits and the underscore, corresponding to the following regular expression: r"[a-zA-Z0-9\_]"

The special sequences consist of "\\" and a character from the following list:

\d Matches any decimal digit; equivalent to the set [0-9].

\D The complement of \d. It matches any non-digit character; equivalent to the set [^0-9].

\s Matches any whitespace character; equivalent to [ \t\n\r\f\v].

\S The complement of \s. It matches any non-whitespace character; equiv. to [^ \t\n\r\f\v].

\w Matches any alphanumeric character; equivalent to [a-zA-Z0-9\_]. With LOCALE, it will match the set [a-zA-Z0-9\_] plus characters defined as letters for the current locale.

\W Matches the complement of \w.

\b Matches the empty string, but only at the start or end of a word.

\B Matches the empty string, but not at the start or end of a word.

\\ Matches a literal backslash.

We used the fact the re.search() returns a match object if it matches and None otherwise. We haven't been interested e.g. in what has been matched. The match object contains a lot of data about what has been matched, positions and so on.

A match object contains the methods group(), span(), start() and end(), as it can be seen in the following application:

**register\_test:**

the register\_test() can register the test function, use it on template:

register\_test('digital', lambda v: type(v) in (int, float))

@register\_test def digital(v): return type(v) in (int, float)

@register\_test('digital') def test\_if\_it\_is\_digital(): return type(v) in (int, float) {{ 9 is digital }

**How to Run Python Scripts Using the Command-Line:**

A Python interactive session will allow you to write a lot of lines of code, but once you close the session, you lose everything you’ve written. That’s why the usual way of writing Python programs is by using plain text files. By convention, those files will use the .py extension. (On Windows systems the extension can also be .pyw.)

Python code files can be created with any plain text editor. If you are new to Python programming, you can try [Sublime Text](https://realpython.com/products/sublime-python/), which is a powerful and easy-to-use editor, but you can use any editor you like.

To keep moving forward in this tutorial, you’ll need to create a test script. Open your favorite text editor and write the following code:

*#!/usr/bin/env python3*

*print('Hello World!')*

Save the file in your working directory with the name hello.py. With the test script ready, you can continue reading.

**\_\_call\_\_ in Python:**

Python has a set of built-in methods and \_\_call\_\_ is one of them. The \_\_call\_\_ method enables Python programmers to write classes where the instances behave like functions and can be called like a function. When the instance is called as a function; if this method is defined, x(arg1, arg2, ...) is a shorthand for x.\_\_call\_\_(arg1, arg2, ...).

object() is shorthand for object.\_\_call\_\_()

**Example 1:**

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| class Example:      def \_\_init\_\_(self):          print("Instance Created")        # Defining \_\_call\_\_ method      def \_\_call\_\_(self):          print("Instance is called via special method")    # Instance created  e = Example()    # \_\_call\_\_ method will be called  e() |

**Output :**

Instance Created

Instance is called via special method

**Example 2:**

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| class Product:      def \_\_init\_\_(self):          print("Instance Created")        # Defining \_\_call\_\_ method      def \_\_call\_\_(self, a, b):          print(a \* b)    # Instance created  ans = Product()    # \_\_call\_\_ method will be called  ans(10, 20) |

**Output :**

Instance Created

200

**ARGs and KARGS:**

<https://www.geeksforgeeks.org/args-kwargs-python/>

**Docopt** module in Python: (To pass the args to the python script):

**docopt** is most commonly used to display the help messages and it is invoked with -h or –help option. You can import and call this module with the following command.

**Best library for Asynchronous SSH:**

“import asyncssh”

<https://medium.com/the-elegant-network/a-tale-of-five-python-ssh-libraries-19cb8b72c914>

**return 0 vs exit(0):**

You shall always use sys.exit(exit\_code)

return 0 will not be seen on system level.

Following (**wrong**) code:

if \_\_name\_\_ == "\_\_main\_\_":

return 0

is wrong and complains on last line, that there is standalone return outside a function

Trying this will not complain, but will not be seen on system level:

def main():

return 0

if \_\_name\_\_ == "\_\_main\_\_":

main()

Correct is:

import sys

def main():

sys.exit(0)

if \_\_name\_\_ == "\_\_main\_\_":

main()

**DocOpt Tool for processing the arguments passed to the script:**

**#!/usr/bin/env python3** *//This is needed to execute a python program as “/.pythonprogram” instead of “python3 python program”*

*\_\_doc\_\_="""*

*Usage:*

*GCPP\_HealthCheck.py [--ns=<kn>] [--tclist=<tl>]*

*Options:*

*--ns=<kn> daases where TCs need to be exected. [default: all]*

*--tclist=<tl> TCs which need to be exected. [default: all]*

*"""*

*def main():*

*args = docopt(\_\_doc\_\_)*

*print("Arguments passed the script are:",args)*

*print("Expected DAAS info are: ", args['--ns'])*

*print("Expected TCs info are: ", args['--tclist'])*

**Formatting the Output:**

import logging

logging.basicConfig(format='%(process)d-%(levelname)s-%(message)s')

logging.warning('This is a Warning')

18472-WARNING-This is a Warning