# [unittest](https://docs.python.org/3/library/unittest.html" \l "module-unittest" \o "unittest: Unit testing framework for Python.) — Unit testing framework

The [**unittest**](https://docs.python.org/3/library/unittest.html#module-unittest) unit testing framework was originally inspired by JUnit and has a similar flavor as major unit testing frameworks in other languages. It supports test automation, sharing of setup and shutdown code for tests, aggregation of tests into collections, and independence of the tests from the reporting framework.

To achieve this, [unittest](https://docs.python.org/3/library/unittest.html#module-unittest) supports some important concepts in an object-oriented way:

**test fixture**

A test fixture represents the preparation needed to perform one or more tests, and any associated cleanup actions. This may involve, for example, creating temporary or proxy databases, directories, or starting a server process.

**test case**

A test case is the individual unit of testing. It checks for a specific response to a particular set of inputs. [unittest](https://docs.python.org/3/library/unittest.html#module-unittest) provides a base class, [TestCase](https://docs.python.org/3/library/unittest.html#unittest.TestCase), which may be used to create new test cases.

**test suite**

A test suite is a collection of test cases, test suites, or both. It is used to aggregate tests that should be executed together.

**test runner**

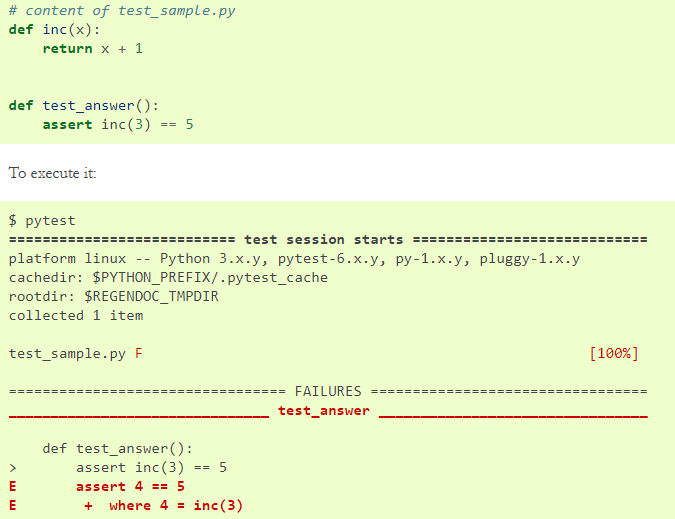
A test runner is a component which orchestrates the execution of tests and provides the outcome to the user. The runner may use a graphical interface, a textual interface, or return a special value to indicate the results of executing the tests.

**Pytest**

* Third –party unittest framework within lighter weight syntax for writing tests

# helps you write better programs

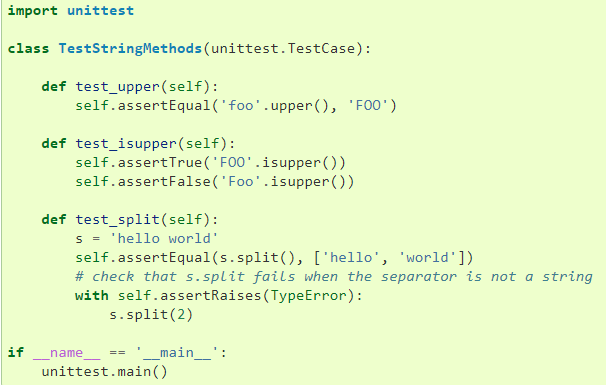
The pytest framework makes it easy to write small tests, yet scales to support complex functional testing for applications and libraries.

An example of a simple test:

## **Basic example**

The [unittest](https://docs.python.org/3/library/unittest.html#module-unittest) module provides a rich set of tools for constructing and running tests. This section demonstrates that a small subset of the tools suffice to meet the needs of most users.

Here is a short script to test three string methods:



A testcase is created by subclassing [unittest.TestCase](https://docs.python.org/3/library/unittest.html#unittest.TestCase). The three individual tests are defined with methods whose names start with the letters test. This naming convention informs the test runner about which methods represent tests.

The crux of each test is a call to [assertEqual()](https://docs.python.org/3/library/unittest.html#unittest.TestCase.assertEqual) to check for an expected result; [assertTrue()](https://docs.python.org/3/library/unittest.html#unittest.TestCase.assertTrue) or [assertFalse()](https://docs.python.org/3/library/unittest.html#unittest.TestCase.assertFalse) to verify a condition; or [assertRaises()](https://docs.python.org/3/library/unittest.html#unittest.TestCase.assertRaises) to verify that a specific exception gets raised. These methods are used instead of the [assert](https://docs.python.org/3/reference/simple_stmts.html#assert) statement so the test runner can accumulate all test results and produce a report.

The [setUp()](https://docs.python.org/3/library/unittest.html#unittest.TestCase.setUp) and [tearDown()](https://docs.python.org/3/library/unittest.html#unittest.TestCase.tearDown) methods allow you to define instructions that will be executed before and after each test method. They are covered in more detail in the section [Organizing test code](https://docs.python.org/3/library/unittest.html#organizing-tests).

The final block shows a simple way to run the tests. [unittest.main()](https://docs.python.org/3/library/unittest.html#unittest.main) provides a command-line interface to the test script. When run from the command line, the above script produces an output that looks like this:

...

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

Ran 3 tests **in** 0.000s

OK

Passing the -v option to your test script will instruct [unittest.main()](https://docs.python.org/3/library/unittest.html#unittest.main) to enable a higher level of verbosity, and produce the following output:

test\_isupper (\_\_main\_\_.TestStringMethods) ... ok

test\_split (\_\_main\_\_.TestStringMethods) ... ok

test\_upper (\_\_main\_\_.TestStringMethods) ... ok

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

Ran 3 tests **in** 0.001s

OK

The above examples show the most commonly used [unittest](https://docs.python.org/3/library/unittest.html#module-unittest) features which are sufficient to meet many everyday testing needs. The remainder of the documentation explores the full feature set from first principles.

## **Command-Line Interface**

The unittest module can be used from the command line to run tests from modules, classes or even individual test methods:

python -m unittest test\_module1 test\_module2

python -m unittest test\_module.TestClass

python -m unittest test\_module.TestClass.test\_method

You can pass in a list with any combination of module names, and fully qualified class or method names.

Test modules can be specified by file path as well:

python -m unittest tests/test\_something.py

This allows you to use the shell filename completion to specify the test module. The file specified must still be importable as a module. The path is converted to a module name by removing the ‘.py’ and converting path separators into ‘.’. If you want to execute a test file that isn’t importable as a module you should execute the file directly instead.

You can run tests with more detail (higher verbosity) by passing in the -v flag:

python -m unittest -v test\_module

When executed without arguments [Test Discovery](https://docs.python.org/3/library/unittest.html#unittest-test-discovery) is started:

python -m unittest

For a list of all the command-line options:

python -m unittest -h

### setUpClass and tearDownClass

These must be implemented as class methods:

import unittest

class Test(unittest.TestCase):

@classmethod

def setUpClass(cls):

cls.\_connection = createExpensiveConnectionObject()

@classmethod

def tearDownClass(cls):

cls.\_connection.destroy()

Tests can be numerous, and their set-up can be repetitive. Luckily, we can factor out set-up code by implementing a method called [setUp()](https://docs.python.org/3/library/unittest.html#unittest.TestCase.setUp), which the testing framework will automatically call for every single test we run:

**import** **unittest**

**class** **WidgetTestCase**(unittest.TestCase):

**def** setUp(self):

self.widget = Widget('The widget')

**def** test\_default\_widget\_size(self):

self.assertEqual(self.widget.size(), (50,50),

'incorrect default size')

**def** test\_widget\_resize(self):

self.widget.resize(100,150)

self.assertEqual(self.widget.size(), (100,150),

'wrong size after resize')

**Note**

The order in which the various tests will be run is determined by sorting the test method names with respect to the built-in ordering for strings.

If the [setUp()](https://docs.python.org/3/library/unittest.html#unittest.TestCase.setUp) method raises an exception while the test is running, the framework will consider the test to have suffered an error, and the test method will not be executed.

Similarly, we can provide a [tearDown()](https://docs.python.org/3/library/unittest.html#unittest.TestCase.tearDown) method that tidies up after the test method has been run:

**import** **unittest**

**class** **WidgetTestCase**(unittest.TestCase):

**def** setUp(self):

self.widget = Widget('The widget')

**def** tearDown(self):

self.widget.dispose()

If [setUp()](https://docs.python.org/3/library/unittest.html#unittest.TestCase.setUp) succeeded, [tearDown()](https://docs.python.org/3/library/unittest.html#unittest.TestCase.tearDown) will be run whether the test method succeeded or not.

Such a working environment for the testing code is called a *test fixture*. A new TestCase instance is created as a unique test fixture used to execute each individual test method. Thus [setUp()](https://docs.python.org/3/library/unittest.html#unittest.TestCase.setUp), [tearDown()](https://docs.python.org/3/library/unittest.html#unittest.TestCase.tearDown), and \_\_init\_\_() will be called once per test.

It is recommended that you use TestCase implementations to group tests together according to the features they test. [unittest](https://docs.python.org/3/library/unittest.html#module-unittest) provides a mechanism for this: the *test suite*, represented by [unittest](https://docs.python.org/3/library/unittest.html#module-unittest)’s [TestSuite](https://docs.python.org/3/library/unittest.html#unittest.TestSuite) class. In most cases, calling [unittest.main()](https://docs.python.org/3/library/unittest.html#unittest.main) will do the right thing and collect all the module’s test cases for you and execute them.

However, should you want to customize the building of your test suite, you can do it yourself:

**def** suite():

suite = unittest.TestSuite()

suite.addTest(WidgetTestCase('test\_default\_widget\_size'))

suite.addTest(WidgetTestCase('test\_widget\_resize'))

**return** suite

**if** \_\_name\_\_ == '\_\_main\_\_':

runner = unittest.TextTestRunner()

runner.run(suite())

You can place the definitions of test cases and test suites in the same modules as the code they are to test (such as widget.py), but there are several advantages to placing the test code in a separate module, such as test\_widget.py:

* The test module can be run standalone from the command line.
* The test code can more easily be separated from shipped code.
* There is less temptation to change test code to fit the code it tests without a good reason.
* Test code should be modified much less frequently than the code it tests.
* Tested code can be refactored more easily.
* Tests for modules written in C must be in separate modules anyway, so why not be consistent?
* If the testing strategy changes, there is no need to change the source code.

# Link: <https://docs.python.org/3/library/unittest.html#setupclass-and-teardownclass>

# Unit testing using Python

# Testing at the function level

# It validate the positive and negative scenario

# Unit tests are the first safety net for catching bugs before they get to the field

# Unit test validate test cases for individual function

# They should be built and run in developer’s development environment

# Unit test should run fast

# Component Testing – Testing is at the library and compiled binary level

# System Testing: Test the external interfaces of a system which is a collection of sub-system.

# A simple Example

# import pytest

# #Production Code

# def str\_len(sstr):

# return len(sstr)

# # A unit test

# def test\_string\_length():

# testStr = '1' #setup

# result = str\_len(testStr) # Action

# assert result ==1 #Assertion

# What is test driven development?

# A process where the developer takes personal responsibility for the quality of their code

# Unit tests are written before the production code

# Don’t write all the tests or production code at once

# Tests and production codes are both written together in small bit of functionalities

# What are some of the benefits of TDD?

# Gives you the confidence to change the code

# Gives you immediate feedback

# Document what code is doing

# Drives good object oriented design

# 

# What is Test Doubles?

# All most all code depends with other parts of the system.

# Those other parts of the system are not always easy to replicate in the unit test environment or would make tests slow if used directly.

# Test doubles are objects that are used in unit tests as replacements to the real production system collaborators

# Types of Test Doubles

# Fake – These objects generally have a simplified functional implementation of a particular interface that is adequate for testing but not for production

# Stub: These objects provides implementations with canned answers that are suitable for the test

# Mocks – These objects are pre-programmed to expect specific calls and parameters and can throw exceptions when necessary

# Mock framework

# Most mock frameworks provide easy ways for automatically creating any of these types of test doubles at run time.

# They provide a fast means for creating mocking expectations for your tests

# Creating mocks objects by hand can be tedious and error prone

# They can be much more efficient than implementing custom mock object of your own creation

# Unittest.mock

# Python mocking framework

# Built in to python version 3.3 and newer

# Needs to be installed for older versions of python with the command “pip install mock”

# 

# 

# 

# 

# 

# Django Development

# The basics of python 3.7

# Pip

# Basic Django project

# Building a personal portfolio with Django

# A text editor to write code, such as Atom

# Basic HTML and JavaScript knowledge

# Functional tests

# Things the user would know and care about

# Order Pizza

# Unit tests

# Things the user would never know about

# Ensure pizza\_description() is working properly

# Python latest version: python-3.9.7

# Pip install django

# Command for creating django project: django-admin startproject testProject

# To start django server: python manage.py runserver

# To create a Django app: python manage.py startapp adoption

# Django App

# A component in a Django project

# A folder with a set of python files

# Each app fits a specific purpose

# Example: Blog, Forum, Wiki,

# 

# 

# Models.py: The models.py file contains the set of models for its Django app

# 

# Models.py: The models.py file contains the set of models for its Django app

# 

# 25 Sep 2021

# Building a personal portfolio with Django

# Section 80.3: Making package executable

# If your package isn't only a library, but has a piece of code that can be used either as a showcase or a standalone application when your package is installed, put that piece of code into \_\_main\_\_.py file.

# Put the \_\_main\_\_.py in the package\_name folder. This way you will be able to run it directly from console:

# python -m package\_name

If there's no \_\_main\_\_.py file available, the package won't run with this command and this error will be printed:

python: No module named package\_name.\_\_main\_\_; 'package\_name' is a package and cannot be directly executed.

**Section 82.1: Install Packages**

To install the latest version of a package named SomePackage:

$ pip install SomePackage

To install a specific version of a package:

$ pip install SomePackage==1.0.4

To specify a minimum version to install for a package:

$ pip install SomePackage>=1.0.4

If commands shows permission denied error on Linux/Unix then use sudo with the commands

**Section 82.2: To list all packages installed using `pip`**

To list installed packages:

$ pip list

# example output

docutils (0.9.1)

Jinja2 (2.6)

Pygments (1.5)

Sphinx (1.1.2)

To list outdated packages, and show the latest version available:

$ pip list --outdated

# example output

docutils (Current: 0.9.1 Latest: 0.10)

Sphinx (Current: 1.1.2 Latest: 1.1.3)

*$ pip install --upgrade SomePackage*

*will upgrade package SomePackage and all its dependencies. Also, pip automatically removes older version of the*

*package before upgrade.*

To upgrade pip itself, do

$ pip install --upgrade pip

on Unix or

$ python -m pip install --upgrade pip

on Windows machines.

**Section 82.4: Uninstall Packages**

To uninstall a package:

$ pip uninstall SomePackage

**Singleton Pattern**:

It is a type of creational pattern which provides a mechanism to have only one and one object of a given type and provides a global point of access.

e.g. Singleton can be used in database operations, where we want database object to maintain data consistency.

**Implementation**

We can implement Singleton Pattern in Python by creating only one instance of Singleton class and serving the same object again. In python, we don’t have any way to create private constructor.

# No. of Python Programs: 1000

Topics needs to be studied:

1. Itertools
2. Python collections
3. [Python Comprehensions](https://goo.gl/6yeo16)
4. Multi threading
5. Data visualization, analysis
6. Socket programming
7. Regular expressions

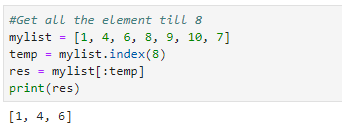
[1] Python | Get elements till particular element in list

|  |
| --- |
| # Python3 code to demonstrate working of  # Get elements till particular element in list  # using index() + list slicing    # initialize list  test\_list = [1, 4, 6, 8, 9, 10, 7]    # printing original list  print("The original list is : " + str(test\_list))    # declaring elements till which elements required  N = 8    # Get elements till particular element in list  # using index() + list slicing  temp = test\_list.index(N)  res = test\_list[:temp]    # printing result  print("Elements till N in list are : " + str(res)) |

Output :

The original list is : [1, 4, 6, 8, 9, 10, 7]

Elements till N in list are : [1, 4, 6]



**Method#2: Using generator**  
This task can also be performed using the generator function which uses yield to get the elements just till the required element and breaks the yields after that element.

|  |
| --- |
| # Python3 code to demonstrate working of  # Get elements till particular element in list  # using generator    # helper function to perform task  def print\_ele(test\_list, val):      for ele in test\_list:          if ele == val:              return          yield ele    # initialize list  test\_list = [1, 4, 6, 8, 9, 10, 7]    # printing original list  print("The original list is : " + str(test\_list))    # declaring elements till which elements required  N = 8    # Get elements till particular element in list  # using generator  res = list(print\_ele(test\_list, N))    # printing result  print("Elements till N in list are : " + str(res)) |

Output :

The original list is : [1, 4, 6, 8, 9, 10, 7]

Elements till N in list are : [1, 4, 6]

# [2] Python | Maximum sum of elements of list in a list of lists

Given lists in a list, find the maximum sum of elements of list in a list of lists.

Examples:

Input : [[1, 2, 3], [4, 5, 6], [10, 11, 12], [7, 8, 9]]

Output : 33

Explanation: sum of all lists in the given list of lists are:

list1 = 6, list2 = 15, list3 = 33, list4 = 24

so the maximum among these is of

Input : [[3, 4, 5], [1, 2, 3], [0, 9, 0]]

Output : 12

|  |
| --- |
| # Python program to find the  # list in a list of lists whose  # sum of elements is the highest  # using traversal    def maximumSum(list1):      maxi = 0        # traversal in the lists      for x in list1:          sum = 0          # traversal in list of lists          for y in x:              sum+= y          maxi = max(sum, maxi)      return maxi    # driver code  list1 = [[1, 2, 3], [4, 5, 6], [10, 11, 12], [7, 8, 9]]  print maximumSum(list1) |

Output:

33

|  |
| --- |
| # Python program to find the  # list in a list of lists whose  # sum of elements is the highest  # using sum and max function and traversal    def maximumSum(list1):      maxi = 0      # traversal      for x in list1:          maxi = max(sum(x), maxi)        return maxi      # driver code  list1 = [[1, 2, 3], [4, 5, 6], [10, 11, 12], [7, 8, 9]]  print maximumSum(list1) |

Output:

33

|  |
| --- |
| # Python program to find the  # list in a list of lists whose  # sum of elements is the highest  # using sum and max function    def maximumSum(list1):      return(sum(max(list1, key = sum)))      # driver code  list1 = [[1, 2, 3], [4, 5, 6], [10, 11, 12], [7, 8, 9]]  print maximumSum(list1) |

Output:

33

# [3] Python | How to get the last element of list

List, being an essential python container is used in day-day programming and also in web-development. Knowledge of its operations is necessary.

Let’s see all the different ways of accessing the last element of a list.

|  |
| --- |
| # Python 3 code to demonstrate  # accessing last element of list  # using naive method    # initializing list  test\_list = [1, 4, 5, 6, 3, 5]    # printing original list  print ("The original list is : " + str(test\_list))    # First naive method  # using loop method to print last element  for i in range(0, len(test\_list)):        if i == (len(test\_list)-1):          print ("The last element of list using loop : "                                    +  str(test\_list[i]))    # Second naive method  # using reverse method to print last element  test\_list.reverse()  print("The last element of list using reverse : "                              +  str(test\_list[0])) |

Output :

The original list is : [1, 4, 5, 6, 3, 5]

The last element of list using loop : 5

The last element of list using reverse : 5

Method #2 : Using [] operator

The last element can be assessed easily if no. of elements in list are already known. There are 2 index in Python that point to last element in list.

* list[ len - 1 ] : Points to last element by definition.
* list[-1] : In python, negative indexing starts from end.

|  |
| --- |
| # Python3 code to demonstrate  # accessing last element of list  # using [] operator    # initializing list  test\_list = [1, 4, 5, 6, 3, 5]    # printing original list  print ("The original list is : " + str(test\_list))    # using len - 1 index to print last list element  print ("The last element using [ len -1 ] is : "             +  str(test\_list[len(test\_list) -1]))    # using -1 index to print last list element  print ("The last element using [ -1 ] is : "                        +  str(test\_list[-1])) |

Output :

The original list is : [1, 4, 5, 6, 3, 5]

The last element using [ len -1 ] is : 5

The last element using [ -1 ] is : 5

**Method #3 : Using**list.pop()

The list.pop() method is used to access the last element of the list. The drawback of this approach is that it also deletes the list last element, hence is only encouraged to use when list is not to be reused.

|  |
| --- |
| # Python3 code to demonstrate  # accessing last element of list  # using list.pop()    # initializing list  test\_list = [1, 4, 5, 6, 3, 5]    # printing original list  print ("The original list is : " + str(test\_list))    # using pop() to print last list element  print ("The last element using pop() is : "                    +  str(test\_list.pop())) |

Output :

The original list is : [1, 4, 5, 6, 3, 5]

The last element using pop() is : 5

**Method #4 : Using**reversed()**+**next()

reversed() coupled with next() can easily be used to get the last element, as like one of the naive method, reversed method returns the reversed ordering of list as an iterator, and next() method prints the next element, in this case last element.

|  |
| --- |
| # Python3 code to demonstrate  # accessing last element of list  # using reversed() + next()    # initializing list  test\_list = [1, 4, 5, 6, 3, 5]    # printing original list  print ("The original list is : " + str(test\_list))    # using reversed() + next() to print last element  print ("The last element using reversed() + next() is : "                          +  str(next(reversed(test\_list)))) |

Output :

The original list is : [1, 4, 5, 6, 3, 5]

The last element using reversed() + next() is : 5

# [5] Python | Find keys with duplicate values in dictionary

Given a dictionary, the task is to find keys with duplicate values. Let’s discuss a few methods for the same.

Method #1: Using Naive approach  
In this method first, we convert dictionary values to keys with the inverse mapping and then find the duplicate keys

|  |
| --- |
| # Python code to demonstrate  # finding duplicate values from a dictionary    # initialising dictionary  ini\_dict = {'a':1, 'b':2, 'c':3, 'd':2}    # printing initial\_dictionary  print("initial\_dictionary", str(ini\_dict))    # finding duplicate values  # from dictionary  # using a naive approach  rev\_dict = {}    for key, value in ini\_dict.items():      rev\_dict.setdefault(value, set()).add(key)    result = [key for key, values in rev\_dict.items()                                if len(values) > 1]    # printing result  print("duplicate values", str(result)) |

Output:

initial\_dictionary {'c': 3, 'b': 2, 'd': 2, 'a': 1}

duplicate values [2]

|  |
| --- |
| # Python code to demonstrate  # finding duplicate values from dictionary    # initialising dictionary  ini\_dict = {'a':1, 'b':2, 'c':3, 'd':2}    # printing initial\_dictionary  print("initial\_dictionary", str(ini\_dict))    # finding duplicate values  # from dictionary using flip  flipped = {}    for key, value in ini\_dict.items():      if value not in flipped:          flipped[value] = [key]      else:          flipped[value].append(key)    # printing result  print("final\_dictionary", str(flipped)) |

Output:

initial\_dictionary {'a': 1, 'c': 3, 'd': 2, 'b': 2}

final\_dictionary {1: ['a'], 2: ['d', 'b'], 3: ['c']}

Method #3: Using chain and set

Suppose you need to find keys having duplicate values.

|  |
| --- |
| # Python code to demonstrate  # finding duplicate values from dictionary  from itertools import chain    # initialising dictionary  ini\_dict = {'a':1, 'b':2, 'c':3, 'd':2}    # printing initial\_dictionary  print("initial\_dictionary", str(ini\_dict))    # finding duplicate values  # from dictionary using set  rev\_dict = {}  for key, value in ini\_dict.items():      rev\_dict.setdefault(value, set()).add(key)      result = set(chain.from\_iterable(           values for key, values in rev\_dict.items()           if len(values) > 1))    # printing result  print("resultant key", str(result)) |

Output:

initial\_dictionary {'b': 2, 'd': 2, 'c': 3, 'a': 1}

resultant key {'d', 'b'}

# [6] Python | Convert a list to dictionary

Given a list, write a Python program to convert the given list to dictionary such that all the odd elements have the key, and even number elements have the value. Since python dictionary is unordered, the output can be in any order.

**Examples:**

**Input :** ['a', 1, 'b', 2, 'c', 3]

**Output :** {'a': 1, 'b': 2, 'c': 3}

**Input :** ['Delhi', 71, 'Mumbai', 42]

**Output :** {'Delhi': 71, 'Mumbai': 42}

**Method #1 :**dict comprehension

To convert a list to dictionary, we can use list comprehension and make a key:value pair of consecutive elements. Finally, typecase the list to *dict* type.

|  |
| --- |
| # Python3 program to Convert a  # list to dictionary    def Convert(lst):      res\_dct = {lst[i]: lst[i + 1] for i in range(0, len(lst), 2)}      return res\_dct    # Driver code  lst = ['a', 1, 'b', 2, 'c', 3]  print(Convert(lst)) |

Output:

{'a': 1, 'b': 2, 'c': 3}

**Method #2 :**Using *zip()* method

First create an iterator, and intialise it to variable ‘it’. Then use *zip* method, to zip keys and values together. Finally typecast it to *dict* type.

|  |
| --- |
| # Python3 program to Convert a  # list to dictionary    def Convert(a):      it = iter(lst)      res\_dct = dict(zip(it, it))      return res\_dct    # Driver code  lst = ['a', 1, 'b', 2, 'c', 3]  print(Convert(lst)) |

Output:

{'c': 3, 'b': 2, 'a': 1}

[6] Q.2. Have you heard of the yield keyword in Python?  
Ans. Yes, I have. This keyword bears the ability to turn any function into a generator. Much like the standard return keyword, but returns a[generator](https://data-flair.training/blogs/python-generators/)object. It is also true that one function may observe multiple yields.

1. >>> def odds(n):
2. odd=[i for i in range(n+1) if i%2!=0]
3. for i in odd:
4. yield i
5. >>> for i in odds(8):
6. print(i)

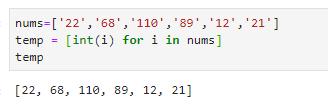
1  
3  
5  
7

[7] Q.3. How will you print the contents of a file?  
Ans.

1. >>> try:
2. with open('tabs.txt','r') as f:
3. print(f.read())
4. except IOError:
5. print("File not found")

[8] Q.4. How will you convert a list into a string?  
Ans. We will use the join() method for this.

1. >>> nums=['one','two','three','four','five','six','seven']
2. >>> s=' '.join(nums)
3. >>> s



[9] Q.5. What will the following code output?  
>>> a=1  
>>> a,b=a+1,a+1  
>>> a,b  
Ans. The output is (2, 2). This code increments the value of a by 1 and assigns it to both a and b. This is because this is a simultaneous declaration. The following code gives us the same:

1. >>> a=1
2. >>> b,a=a+1,a+1
3. >>> a,b

(2, 2)

[10]Explain different ways to create an empty NumPy array in Python.  
Ans. We’ll talk about two methods to [create NumPy](https://data-flair.training/blogs/python-numpy-tutorial/) array-

1. First method-
2. >>> import numpy
3. >>> numpy.array([])

array([], dtype=float64)

1. Second method-
2. >>> numpy.empty(shape=(0,0))

array([], shape=(0, 0), dtype=float64)

[11] How will you remove a duplicate element from a list?  
Ans. We can turn it into a set to do that.

1. >>> list=[1,2,1,3,4,2]
2. >>> set(list)

{1, 2, 3, 4}

[12] How many types of objects does Python support?  
Ans. [Objects in Python](https://data-flair.training/blogs/python-object/) are mutable and immutable. Let’s talk about these.

* Immutable objects- Those which do not let us modify their contents. Examples of these will be tuples, booleans, strings, integers, floats, and complexes. Iterations on such objects are faster.

1. >>> tuple=(1,2,4)
2. >>> tuple

(1, 2, 4)

1. >>> 2+4j

(2+4j)

* Mutable objects- Those that let you modify their contents. Examples of these are lists, sets, and dicts. Iterations on such objects are slower.

1. >>> [2,4,9]

[2, 4, 9]

1. >>> dict1={1:1,2:2}
2. >>> dict1

{1: 1, 2: 2}  
While two equal immutable objects’ reference variables share the same address, it is possible to create two mutable objects with the same content.

[13] Create a new list to convert the following list of number strings to a list of numbers.  
nums=[‘22’,’68’,’110’,’89’,’31’,’12’]

Ans. We will use the int() function with a list comprehension to convert these strings into integers and put them in a list.

1. >>> [int(i) for i in nums]

[22, 68, 110, 89, 31, 12]

[14] **What is the MRO in Python?**  
**Ans.**[MRO stands for Method Resolution Order](https://data-flair.training/blogs/python-multiple-inheritance/). Talking of multiple inheritances, whenever we search for an attribute in a class, Python first searches in the current class. If found, its search is satiated. If not, it moves to the parent class. It follows an approach that is left-to-right and depth-first. It goes Child, Mother, Father, Object.  
We can call this order a linearization of the class Child; the set of rules applied are the Method Resolution Order (MRO). We can borrow the \_\_mro\_\_ attribute or the mro() method to get this.

Satiate: to completely satisfy a desire or need

[15] What is the best code you can write to swap two numbers?  
Ans. I can perform the swapping in one statement.

1. >>> a,b=b,a

Here’s the entire code, though-

1. >>> a,b=2,3
2. >>> a,b=b,a
3. >>> a,b

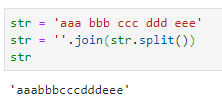
(3, 2)

[16] **Explain the problem with the following piece of code-**  
>>> def func(n=[]):  
 #playing around  
 pass  
>>> func([1,2,3])  
>>> func()  
>>> n  
**Ans.** The request for n raises a NameError. This is since n is a variable local to func and we cannot access it elsewhere. It is also true that Python only evaluates default parameter values once; every invocation shares the default value. If one invocation modifies it, that is what another gets. This means you should only ever use primitives, strings, and tuples as default parameters, not mutable objects.

[17] Can you remove the whitespaces from the string “aaa bbb ccc ddd eee”?

Ans. I can think of two ways to do this.

1. Using join-



1. >>> s='aaa bbb ccc ddd eee'
2. >>> s1=''.join(s.split())
3. >>> s1

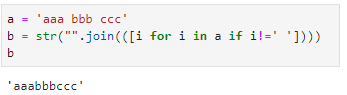
‘aaabbbcccdddeee’

1. Using a list comprehension-

a = 'aaa bbb ccc'

b = str("".join(([i for i in a if i!=' '])))

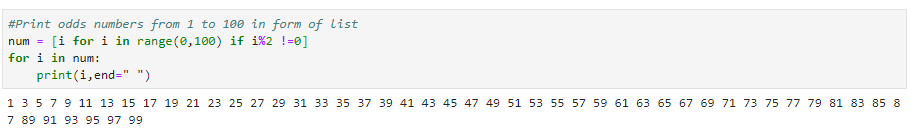
b



1. >>> s='aaa bbb ccc ddd eee'
2. >>> s1=str(''.join(([i for i in s if i!=' '])))
3. >>> s1

‘aaabbbcccdddeee’

[17] Print odds numbers from 1 to 1000 in form of list



def odds(n):

odd = [i for i in range(n+1) if i%2!= 0]

print(odd)

odds(1000)

[18]  How do you get the current working directory using Python?  
Ans. Working on software with Python, you may need to read and write files from various directories. To find out which directory we’re presently working under, we can borrow the getcwd() method from the os module.

1. >>> import os
2. >>> os.getcwd()

[19] What is tuple unpacking?  
Ans. Suppose we have a tuple nums=(1,2,3). We can unpack its values into the variables a, b, and c. Here’s how:  
[Have a look at Python tuple](https://data-flair.training/blogs/python-tuples-syntax-examples/)

1. >>> nums=(1,2,3)
2. >>> a,b,c=nums
3. >>> a

1

1. >>> b

2

1. >>> c

3

[20] What does the following code give us?  
>>> b=(1)  
Ans. Not a tuple. This gives us a plain integer.

1. >>> type(b)

<class ‘int’>  
To let it be a tuple, we can declare so explicitly with a comma after 1:

1. >>> b=(1,)
2. >>> type(b)

<class ‘tuple’>

[21]Why do we need to overload operators?

Ans. To compare two objects, we can[overload operators in Python](https://data-flair.training/blogs/python-operator-overloading/). We understand 3>2. But what is orange>apple? Let’s compare apples and oranges now.

1. >>> class fruit:
2. def \_\_init\_\_(self,type,size):
3. self.type='fruit'
4. self.type=type
5. self.size=size
6. def \_\_gt\_\_(self,other):
7. if self.size>other.size:
8. return True
9. return False
10. >>> orange=fruit('orange',7)
11. >>> apple=fruit('apple',8)
12. >>> apple>orange

True

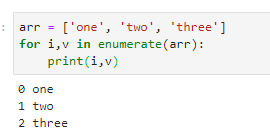
1. >>> orange>apple

False

[22] What is the enumerate() function in Python?

Ans. enumerate() iterates through a sequence and extracts the index position and its corresponding value too.  
Let’s take an example.

1. >>> for i,v in enumerate(['Python','C++','Scala']):
2. print(i,v)



[23]

[24] Is there a way to remove the last object from a list?

Yes, there is. Try running the following piece of code-

1. >>> list=[1,2,3,4,5]
2. >>> list.pop(-1)

[25] **Can you explain the filter(), map(), and reduce() functions?**

Let’s see these[Python Functions](https://data-flair.training/blogs/python-functions/).

* **filter()-** This function lets us keep the values that satisfy some conditional logic. Let’s take an example.

1. >>> set(filter(lambda x:x>4, range(7)))

**{5, 6}**

This filters in the elements from 0 to 6 that are greater than the number 4.

* **map()-** This function applies a function to each element in the iterable.

1. >>> set(map(lambda x:x\*\*3, range(7)))

**{0, 1, 64, 8, 216, 27, 125}**

This calculates the cube for each element in the range 0 to 6 and stores them in a set.

* **reduce()-** This function reduces a sequence pair-wise, repeatedly until we arrive at a single value.

from functools import reduce

reduce(lambda x,y:y-x, [1,2,3,4,5])

**3**

Let’s understand this:

2-1=1

3-1=2

4-2=2

5-2=3

Hence, 3.

[24] Why do we need the \_\_init\_\_() function in classes? What else helps?

\_\_init\_\_() is what we need to initialize a class when we initiate it. Let’s take an example.

1. >>> class orange:
2. def \_\_init\_\_(self):
3. self.color='orange'
4. self.type='citrus'
5. def setsize(self,size):
6. self.size=size
7. def show(self):
8. print(f"color: {self.color}, type: {self.type}, size: {self.size}")
9. >>> o=orange()
10. >>> o.setsize(7)
11. >>> o.show()

color: orange, type: citrus, size: 7

In this code, we see that it is necessary to pass the parameter ‘self’ to tell Python it has to work with this object.

[25]Explain the output of the following piece of code-

1. >>> tuple=(123,'John')
2. >>> tuple\*=2
3. >>> tuple

(123, ‘John’, 123, ‘John’)

In this code, we multiply the tuple by 2. This duplicates its contents, hence, giving us (123, ‘John’, 123, ‘John’). We can also do this to strings:

1. >>> 'ba'+'na'\*2

‘banana’

[26] **How do you get all values from a Python dictionary?**

We saw previously, to get [all keys from a dictionary](https://data-flair.training/blogs/python-dictionaries/), we make a call to the keys() method. Similarly, for values, we use the method values().

1. >>> 'd' in {'a':1,'b':2,'c':3,'d':4}.values()

**False**

1. >>> 4 in {'a':1,'b':2,'c':3,'d':4}.values()

**True**

[26] How will you convert an integer to a Unicode character?

This is simple. All we need is the chr(x) built-in function. See how.

1. >>> chr(52)

‘4’

1. >>> chr(49)

‘1’

1. >>> chr(67)

‘C’

[26]In one line, show us how you’ll get the max alphabetical character from a string.

For this, we’ll simply use the max function.

1. >>> max('flyiNg')

‘y’

The following are the ASCII values for all the letters of this string-

f- 102

l- 108

y- 121

i- 105

N- 78

g- 103

[27] Will the do-while loop work if you don’t end it with a semicolon?

Trick question! Python does not support an intrinsic do-while loop. Secondly, to terminate do-while loops is a necessity for languages like C++.

[28] What if you want to toggle case for a Python string?

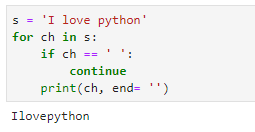
We have the swapcase() method from the str class to do just that.

1. >>> 'AyuShi'.swapcase()

‘aYUsHI’

[29]

[30]Write code to print everything in the string except the spaces.



[31]Now, print this string five times in a row.

1. >>> for i in range(6):
2. print(s)

[32] Is del the same as remove()? What are they?

del and remove() are methods on lists/ ways to eliminate elements.

1. >>> list=[3,4,5,6,7]
2. >>> del list[3]
3. >>> list

[3, 4, 5, 7]

1. >>> list.remove(5)
2. >>> list

[3, 4, 7]

While del lets us delete an element at a certain index, remove() lets us remove an element by its value.

[33] What is the purpose of bytes() in Python?

bytes() is a built-in function in Python that returns an immutable bytes object. Let’s take an example.

1. >>> bytes([2,4,8])

b’\x02\x04\x08′

1. >>> bytes(5)

b’\x00\x00\x00\x00\x00′

1. >>> bytes('world','utf-8')

b’world’

[34] **Explain, in brief, the uses of the modules sqlite3, ctypes, pickle, traceback, and itertools.**

* **sqlite3-** Helps with handling databases of type SQLite
* **ctypes-** Lets create and manipulate C data types in Python
* **pickle-** Lets put any data structure to external files
* **traceback-** Allows extraction, formatting, and printing of stack traces
* [itertools](https://data-flair.training/blogs/python-itertools-tutorial/)**–** Supports working with permutations, combinations, and other useful [iterables](https://data-flair.training/blogs/python-iterables/)**.**

[35] **What is speech\_recognition? Does this ship with Python by default?**

[Speech\_recognition](https://data-flair.training/blogs/python-speech-recognition-ai/) is a library for performing the task of recognizing speech with Python. This forms an integral part of AI. No, this does not ship with Python by default. We must download it from the PyPI and install it manually using pip.

[36] What will the following code output?  
>>> word=’abcdefghij’  
>>> word[:3]+word[3:]

The output is ‘abcdefghij’. The first slice gives us ‘abc’, the next gives us ‘defghij’.

[37]Optionally, what statements can you put under a try-except block?

We have two of those:

* else- To run a piece of code when the try-block doesn’t create an exception.
* finally- To execute some piece of code regardless of whether there is an exception.

1. >>> try:
2. print("Hello")
3. except:
4. print("Sorry")
5. else:
6. print("Oh then")
7. finally:
8. print("Bye")

Hello  
Oh then  
Bye

[38]Differentiate between split(), sub(), and subn() methods of the re module.

Ans. The re module is what we have for processing regular expressions with Python. Let’s talk about the three methods we mentioned-

* split()- This makes use of a regex pattern to split a string into a list
* sub()- This looks for all substrings where the regex pattern matches, and replaces them with a different string
* subn()- Like sub(), this returns the new string and the number of replacements made

[39] **Differentiate between Django, Pyramid, and Flask.**

**Ans.** These are three major[frameworks in Python](https://data-flair.training/blogs/python-web-framework/). Here are the differences:

* We can also use [Django](https://data-flair.training/blogs/python-django-tutorial/) for larger applications. It includes an ORM.
* Flask is a microframework for a small platform with simpler requirements. It is ready to use and you must use external libraries.
* The pyramid is for larger applications. It is flexible and you can choose the database, the URL structure, the templating style, and much more. It is also heavily configurable.

[40] **Explain the Inheritance Styles in Django.**

**Ans.** Talking on [inheritance](https://data-flair.training/blogs/python-multiple-inheritance/) styles, we have three possible-

* **Abstract Base Classes-** For the parent class to hold information so we don’t have to type it out for each child model
* **Multi-table Inheritance-** For subclassing an existing model and letting each model have its own database
* **Proxy Models-** For modifying the model’s Python-level behavior without having to change its fields

[41]Explain Python List Comprehension.

The [list comprehension](https://data-flair.training/blogs/python-list-comprehension/) is a way to declare a list in one line of code. Let’s take a look at one such example.

1. >>> [i for i in range(1,11,2)]

[1, 3, 5, 7, 9]

1. >>> [i\*2 for i in range(1,11,2)]

[2, 6, 10, 14, 18]

[42]How will you locally save an image using its URL address?

Ans. For this, we use the urllib module.

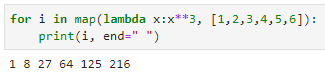
1. >>> import urllib.request
2. >>> urllib.request.urlretrieve('https://yt3.ggpht.com/a-/ACSszfE2YYTfvXCIVk4NjJdDfFSkSVrLBlalZwYsoA=s900-mo-c-c0xffffffff-rj-k-no','dataflair.jpg')

(‘dataflair.jpg’, <http.client.HTTPMessage object at 0x02E90770>)

You can then get to your Python’s location and confirm this.

[43]What does the map() function do?

Ans. map() executes the function we pass to it as the first argument; it does so on all elements of the iterable in the second argument. Let’s take an example, shall we?



1. >>> for i in map(lambda i:i\*\*3, (2,3,7)):
2. print(i)

8  
27  
343

This gives us the cubes of the values 2, 3, and 7.

[44] How will you share global variables across modules?

Ans. To do this for modules within a single program, we create a special module, then import the config module in all modules of our application. This lets the module be global to all modules.

[45]What is the process to calculate percentiles with NumPy?

Ans. Refer to the code below.

1. >>> import numpy as np
2. >>> arr=np.array([1,2,3,4,5])
3. >>> p=np.percentile(arr,50)
4. >>> p

3.0

[46] **What is Flask- WTF? Explain its features.**

**Ans.**[Flask](https://data-flair.training/blogs/python-flask/)-WTF supports simple integration with WTForms. It has the following features-

* Integration with wtforms
* Global csrf protection
* Recaptcha supporting
* Internationalization integration
* Secure form with csrf token
* File upload compatible with Flask uploads

[47] **How is NumPy different from SciPy?**

**Ans.** We have so far seen them used together. But they have subtle differences:

* [**SciPy**](https://data-flair.training/blogs/scipy-tutorial/) encompasses most new features
* NumPy does hold some linear algebra functions
* SciPy holds more fully-featured versions of the linear algebra modules and other numerical algorithms
* [**NumPy**](https://data-flair.training/blogs/python-numpy-tutorial/)has compatibility as one of its essential goals; it attempts to retain all features supported by any of its predecessors
* NumPy holds the array data type and some basic operations: indexing, sorting, reshaping, and more

[48] What is the Dogpile effect?

Ans. In case the cache expires, what happens when a client hits a website with multiple requests is what we call the dogpile effect. To avoid this, we can use a semaphore lock. When the value expires, the first process acquires the lock and then starts to generate the new value.

[49]How do you insert an object at a given index in Python?

Ans. Let’s build a list first.

1. >>> a=[1,2,4]

Now, we use the method insert. The first argument is the index at which to insert, the second is the value to insert.

1. >>> a.insert(2,3)
2. >>> a

[1, 2, 3, 4]

[50]And how do you reverse a list?

Ans. Using the reverse() method.

1. >>> a.reverse()
2. >>> a

[4, 3, 2, 1]

[51] **What is a Python module?**

**Ans.** A [module](https://data-flair.training/blogs/python-modules/)is a script in Python that defines import statements, functions, classes, and variables. It also holds runnable Python code. ZIP files and DLL files can be modules too. The module holds its name as a string that is in a global variable.

[52]What is the with statement in Python?

Ans. Using the with statement, we open files, process data in files, and even close them without having to make a call to the close() method. This makes exception-handling simpler with the cleanup activities. Here’s a demonstration-

1. >>> with open('data.txt') as data:
2. #processing statements

[53]What are the different file-processing modes with Python?

Ans. We have the following modes-

* read-only – ‘r’
* write-only – ‘w’
* read-write – ‘rw’
* append – ‘a’

We can open a text file with the option ‘t’. So to open a text file to read it, we can use the mode ‘rt’. Similarly, for binary files, we use ‘b’.

[54] **What makes Python object-oriented?**

**Ans.** Python is object-oriented because it follows the O[bject-Oriented programming](https://data-flair.training/blogs/python-classes/) paradigm. This is a paradigm that revolves around classes and their instances (objects). With this kind of programming, we have the following features:

Encapsulation

Abstraction

Inheritance

Polymorphism

Data hiding

[55] Does Python support interfaces like Java does?

Ans. No. However, Abstract Base Classes (ABCs) are a feature from the abcmodule that let us declare what methods subclasses should implement. Python supports this module since version 2.7.

[56] Explain the output of the following piece of code-

x=[‘ab’,’cd’]

print(len(map(list,x)))

Ans. This actually gives us an error- a TypeError. This is because map() has no len() attribute in their dir().

[57] So what is the output of the following piece of code?

x=[‘ab’,’cd’]

print(len(list(map(list,x))))

Ans. This outputs 2 because the length of each string is 2.

What is the output of the below code snippet ?

x =['ab', 'cd']

length = map(tuple,['ab', 'cd'])

print(list(length))

[58]Explain garbage collection with Python.

Ans. The following points are worth nothing for the garbage collector with CPython-

* Python maintains a count of how many references there are to each object in memory
* When a reference count drops to zero, it means the object is dead and Python can free the memory it allocated to that object
* The garbage collector looks for reference cycles and cleans them up
* Python uses heuristics to speed up garbage collection
* Recently created objects might as well be dead
* The garbage collector assigns generations to each object as it is created
* It deals with the younger generations first.

[59] **How is Python different from Java?**

**Ans.** Following is the[comparison of Python vs Java](https://data-flair.training/blogs/python-vs-java/)–

**a.** Java is faster than Python

**b.** Python mandates indentation. Java needs braces.

**c.** Python is dynamically-typed; Java is statically typed.

**d.** Python is simple and concise; Java is verbose

**e.** Python is interpreted

**f.** Java is platform-independent

**g.** Java has stronger database-access with JDBC

[60] How do we execute Python?

Ans. Python files first compile to bytecode. Then, the host executes them.

[61] Explain try, raise, and finally.

Ans. These are the keywords we use with exception-handling. We put risky code under a try block, use the raise statement to explicitly raise an error, and use the finally block to put code that we want to execute anyway.

[62] What happens if we do not handle an error in the except block?

Ans. If we don’t do this, the program terminates. Then, it sends an execution trace to sys.stderr.

[63]How does a function return values?

Ans. A function uses the ‘return’ keyword to return a value. Take a look:

1. >>> def add(a,b):
2. return a+b

[64]What does the following code output?

1. >>> def extendList(val, list=[]):
2. list.append(val)
3. return list
4. >>> list1 = extendList(10)
5. >>> list2 = extendList(123,[])
6. >>> list3 = extendList('a')
7. >>> list1,list2,list3

Ans. ([10, ‘a’], [123], [10, ‘a’])

You’d expect the output to be something like this:

([10],[123],[‘a’])

Well, this is because the list argument does not initialize to its default value ([]) every time we make a call to the function. Once we define the function, it creates a new list. Then, whenever we call it again without a list argument, it uses the same list. This is because it calculates the expressions in the default arguments when we define the function, not when we call it.

[65] **How many arguments can the range() function take?**

**Ans.** The[range() function in Python](https://data-flair.training/blogs/range-function-in-python/) can take up to 3 arguments. Let’s see this one by one.

**a. One argument**

When we pass only one argument, it takes it as the stop value. Here, the start value is 0, and the step value is +1.

1. >>> list(range(5))

[0, 1, 2, 3, 4]

1. >>> list(range(-5))

[]

1. >>> list(range(0))

[] **b. Two arguments**  
When we pass two arguments, the first one is the start value, and the second is the stop value.

1. >>> list(range(2,7))

[2, 3, 4, 5, 6]

1. >>> list(range(7,2))

[]

1. >>> list(range(-3,4))

[-3, -2, -1, 0, 1, 2, 3]

**c. Three arguments**

Here, the first argument is the start value, the second is the stop value, and the third is the step value.

1. >>> list(range(2,9,2))

[2, 4, 6, 8]

1. >>> list(range(9,2,-1))

[9, 8, 7, 6, 5, 4, 3]

[66] **Does Python have a switch-case statement?**

[But in Python, we do not have a switch-case statement](https://data-flair.training/blogs/python-switch-case/). Here, you may write a switch function to use. Else, you may use a set of if-elif-else statements. To implement a function for this, we may use a dictionary.

1. >>> def switch(choice):
2. switcher={
3. 'Ayushi':'Monday',
4. 'Megha':'Tuesday',
5. print(switcher.get(choice,'Hi, user'))

return

1. >>> switch('Megha')

Tuesday

1. >>> switch('Ayushi')

Monday

1. >>> switch('Ruchi')

Hi, user

Here, the get() method returns the value of the key. When no key matches, the default value (the second argument) is returned.

[67] **How do you debug a program in Python? Answer in brief.**

**Ans.** To debug a Python program, we use the pdb module. This is the [Python debugger](https://data-flair.training/blogs/python-debugger/); we will discuss it in a tutorial soon. If we start a program using pdb, it will let us step through the code.

**List some pdb commands.**

Some [pdb commands](https://data-flair.training/blogs/python-os-module/) include-

<b> — Add breakpoint

<c> — Resume execution

<s> — Debug step by step

<n> — Move to next line

<l> — List source code

<p> — Print an expression

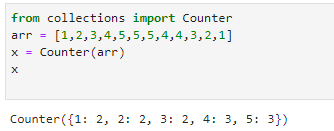
[68] **What is a Counter in Python?**

**Ans.** The function Counter() from the module ‘collections’. It counts the number of occurrences of the elements of a container.

1. >>> from collections import Counter
2. >>> **Counter**([1,3,2,1,4,2,1,3,1])

Counter({1: 4, 3: 2, 2: 2, 4: 1})

Python provides us with a range of ways and methods to work with a Counter. Read[**Python Counter**](https://data-flair.training/blogs/python-counter/).



[69] **What is NumPy? Is it better than a list?**

**Ans.** NumPy, a Python package, has made its place in the world of scientific computing. It can deal with large data sizes, and also has a powerful N-dimensional array object along with a set of advanced functions.

Yes, a NumPy array is better than a [**Python list**](https://data-flair.training/blogs/python-lists-examples/). This is in the following ways:

1. It is more compact.
2. It is more convenient.
3. It Is more efficiently.
4. It is easier to read and write items with NumPy.

[70] **What is PEP 8?**

**Ans.** PEP 8 is a coding convention that lets us write more readable code. In other words, it is a set of recommendations.

[71] **What is pickling and unpickling?**

**Ans.** To create portable serialized representations of [**Python objects**](https://data-flair.training/blogs/python-object/), we have the module ‘pickle’. It accepts a Python object (remember, everything in Python is an object). It then converts it into a string representation and uses the dump() function to dump it into a file. We call this pickling. In contrast, retrieving objects from this stored string representation is termed ‘unpickling’.

[72] **So, then, what is the global keyword?**

**Ans.** Like we saw in the previous question, the global keyword lets us deal with, inside any scope, the global version of a variable.

The problem:

1. >>> a=7
2. >>> def **func**():
3. **print**(a)
4. a+=1
5. **print**(a)

The solution:

1. >>> a=7
2. >>> def **func**():
3. global a
4. **print**(a)
5. a+=1
6. **print**(a)
7. >>> **func**()

7  
8

[73]How would you make a Python script executable on Unix?

Ans. For this to happen, two conditions must be met:

1. The script file’s mode must be executable
2. The first line must begin with a hash(#). An  example of this will be: #!/usr/local/bin/python

[74] What functions or methods will you use to delete a file in Python?

Ans. For this, we may use remove() or unlink().

1. >>> import os
2. >>> os.chdir('C:\\Users\\lifei\\Desktop')
3. >>> os.remove('try.py')
4. >>>

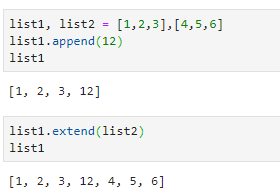
# When we go and check our Desktop, the file is gone. Let’s go make it again so we can delete it again using unlink().

1. >>> os.unlink('try.py')
2. >>>

Both [functions](https://data-flair.training/blogs/python-built-in-functions/) are the same, but unlink is the traditional Unix name for it.

[75]Differentiate between the append() and extend() methods of a list.

Ans. The methods append() and extend() work on lists. While append() adds an element to the end of the list, extend adds another list to the end of a list.



[76] Which methods/functions do we use to determine the type of instance and inheritance?

Ans. Here, we talk about three methods/functions- type(), isinstance(), and issubclass().

a. type()

This tells us the type of object we’re working with.

1. >>> type(3)

<class ‘int’>

1. >>> type(False)

<class ‘bool’>

1. >>> type(lambda :print("Hi"))

<class ‘function’>

1. >>> type(type)

<class ‘type’>

b. isinstance()

This takes in two arguments- a value and a type. If the value is of the kind of the specified type, it returns True. Else, it returns False.

1. >>> isinstance(3,int)

True

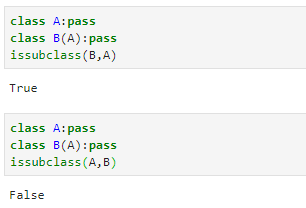
1. >>> isinstance((1),tuple)

False

1. >>> isinstance((1,),tuple)

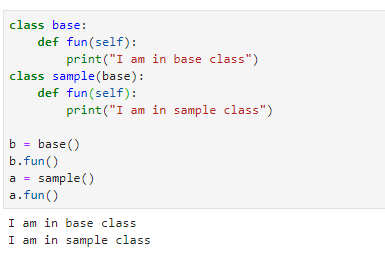
True  
c. issubclass()

This takes two classes as arguments. If the first one inherits from the second, it returns True. Else, it returns False.



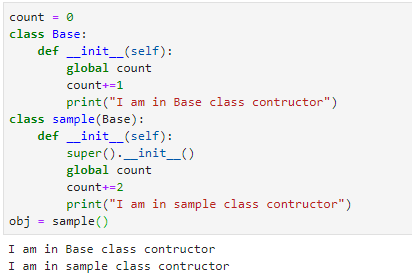
[77]What do you mean by overriding methods?

Ans. Suppose class B inherits from class A. Both have the method sayhello()- to each, their own version. B overrides the sayhello() of class A. So, when we create an object of class B, it calls the version that class B has.



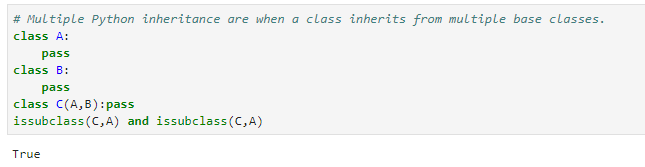
### [78] Single Inheritance in Python

A single Python inheritance is when a single class inherits from a class.



### [79] Python Multiple Inheritance

Multiple Python inheritance are when a class inherits from multiple base classes.



### [80] Multilevel Inheritance in Python

When one class inherits from another, which in turn inherits from another, it is multilevel python inheritance.

1. >>> class A:
2. x=1
3. >>> class B(A):
4. pass
5. >>> class C(B):
6. pass
7. >>> cobj=C()
8. >>> cobj.x

1

### [81] Hierarchical Inheritance in Python

When more than one class inherits from a class, it is hierarchical Python inheritance.

1. >>> class A:
2. pass
3. >>> class B(A):
4. pass
5. >>> class C(A):
6. pass
7. >>> issubclass(B,A) and issubclass(C,A)

True

### [82] Hybrid Inheritance in Python

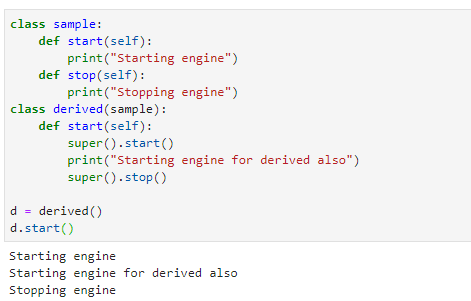
Hybrid Python inheritance is a combination of any two kinds of inheritance.

1. >>> class A:
2. x=1
3. >>> class B(A):
4. pass
5. >>> class C(A):
6. pass
7. >>> class D(B,C):
8. pass
9. >>> dobj=D()
10. >>> dobj.x

1

## [83] Python Inheritance Super Function – Super()

With inheritance, the super() [**function in python**](https://data-flair.training/blogs/python-functions/)actually comes in quite handy. It allows us to call a method from the parent class. Let’s define a new class for this.



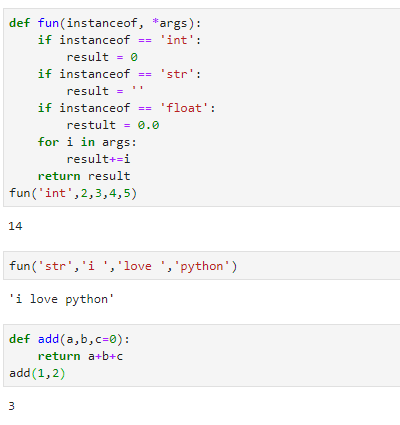
## [84] Python Override Method

A subclass may change the functionality of a [**Python method**](https://data-flair.training/blogs/methods-in-python-programming/) in the superclass. It does so by redefining it. This is termed python method overriding. Lets see this Python Method Overriding Example.

1. >>> class A:
2. def sayhi(self):
3. print("I'm in A")
4. >>> class B(A):
5. def sayhi(self):
6. print("I'm in B")
7. >>> bobj=B()
8. >>> bobj.sayhi()

I’m in B

## [85] Python Method Overloading



## [86] Python Inheritance Syntax

To make a class inherit from another, we apply the name of the base class in parentheses to the derived class’ definition.

1. >>> class Person:
2. pass
3. >>> class Student(Person):
4. pass
5. >>> issubclass(Student,Person)

True  
Here, class Student inherits from class Person. Here, since we only want to focus on the [**python syntax**](https://data-flair.training/blogs/python-syntax-semantics/), we use the ‘pass’ statement in the bodies of the classes. Also, we use the function issubclass() to confirm that student is a subclass of person.

[87] **What data types does Python support?**

This is the most basic python interview question.

Python provides us with five kinds of data types:

* **Numbers-** [Numbers](https://data-flair.training/blogs/python-number-types-conversion/) use to hold numerical values.

1. >>> a=7.0
2. >>>

* **Strings-** A [string](https://data-flair.training/blogs/python-strings/) is a sequence of characters.  We declare it using single or double quotes.

1. >>> title="Ayushi's Book"

* **Lists-** A [list](https://data-flair.training/blogs/python-list-comprehension/) is an ordered collection of values, and we declare it using square brackets.

1. >>> colors=['red','green','blue']
2. >>> type(colors)

**<class ‘list’>**

* **Tuples-** A [tuple](https://data-flair.training/blogs/python-tuples-syntax-examples/), like a list, is an ordered collection of values. The difference. However, is that a tuple is immutable. This means that we cannot change a value in it.

1. >>> name=('Ayushi','Sharma')
2. >>> name[0]='Avery'

*Traceback (most recent call last):*  
*File “<pyshell#129>”, line 1, in <module>*  
*name[0]=’Avery’*  
*TypeError: ‘tuple’ object does not support item assignment*

1. **Dictionary-** A [dictionary](https://data-flair.training/blogs/python-dictionaries/) is a data structure that holds key-value pairs. We declare it using curly braces.
2. >>> squares={1:1,2:4,3:9,4:16,5:25}
3. >>> type(squares)

**<class ‘dict’>**

1. >>> type({})

**<class ‘dict’>**  
We can also use a dictionary comprehension:

1. >>> squares={x:x\*\*2 for x in range(1,6)}
2. >>> squares

{1: 1, 2: 4, 3: 9, 4: 16, 5: 25}

[88]What is a docstring?

A docstring is a documentation string that we use to explain what a construct does. We place it as the first thing under a function, class, or a method, to describe what it does. We declare a docstring using three sets of single or double quotes.

1. >>> def sayhi():
2. """
3. The function prints Hi
4. """
5. print("Hi")
6. >>> sayhi()

Hi

To get a function’s docstring, we use its \_\_doc\_\_ attribute.

1. >>> sayhi.\_\_doc\_\_

‘\n\tThis function prints Hi\n\t’

A docstring, unlike a comment, is retained at runtime.

[89] **What is slicing?**

These are the types of basic Python interview questions for freshers.  
Slicing is a technique that allows us to retrieve only a part of a [list](https://data-flair.training/blogs/python-list-comprehension/), tuple, or string. For this, we use the slicing operator [].

1. >>> (1,2,3,4,5)[2:4]

(3, 4)

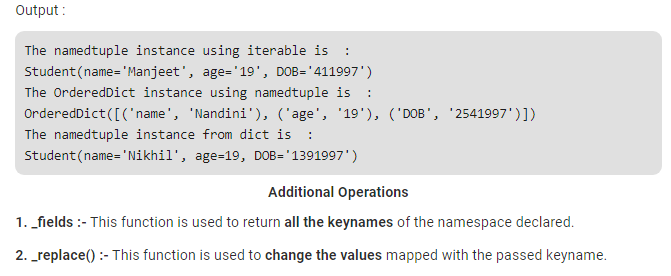
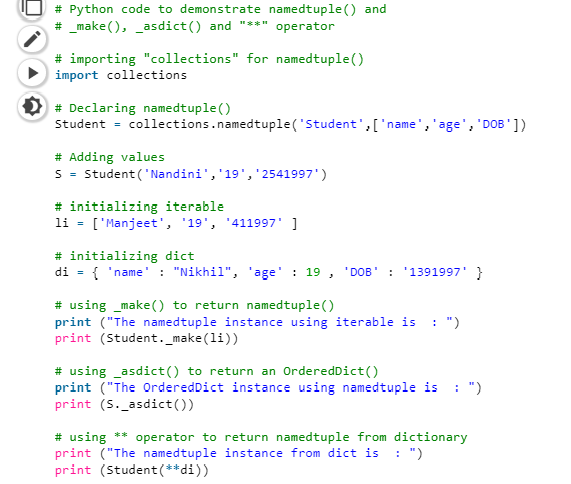
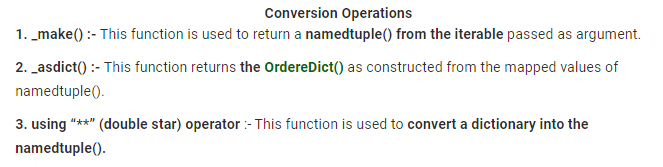
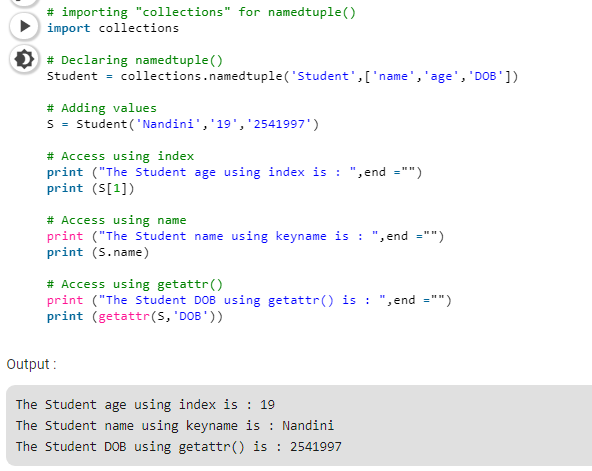
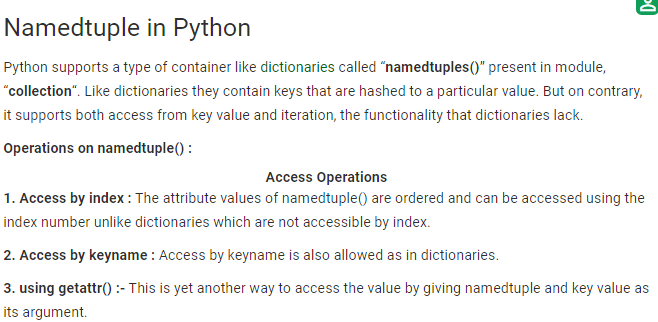
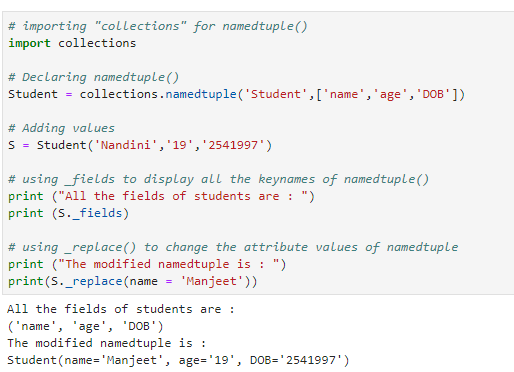
1. >>> [7,6,8,5,9][2:]

[8, 5, 9]

1. >>> 'Hello'[:-1]

‘Hell’

[90]What is a namedtuple?

[91] **How would you convert a string into an int in Python?**

If a [string](https://data-flair.training/blogs/python-strings/) contains only numerical characters, you can convert it into an integer using the int() function.

1. >>> int('227')

227

Let’s check the types:

1. >>> type('227')

**<class ‘str’>**

1. >>> type(int('227'))

**<class ‘int’>**

[92] **How do you take input in Python?**

For taking input from user, we have the function input(). In Python 2, we had another function raw\_input().

The[input() function](https://data-flair.training/blogs/python-built-in-functions/)takes, as an argument, the text to be displayed for the task:

1. >>> a=input('Enter a number')

Enter a number7

But if you have paid attention, you know that it takes input in the form of a string.

1. >>> type(a)

**<class ‘str’>**

Multiplying this by 2 gives us this:

1. >>> a\*=2
2. >>> a

’77’

So, what if we need to work on an integer instead?

We use the int() function for this.

1. >>> a=int(input('Enter a number'))

Enter a number7

Now when we multiply it by 2, we get this:

1. >>> a\*=2
2. >>> a

14

[93] **What is a frozen set in Python?**

Answer these type of Python Interview Questions with Examples.

First, let’s discuss what a set is. A set is a collection of items, where there cannot be any duplicates. A set is also unordered.

1. >>> myset={1,3,2,2}
2. >>> myset

{1, 2, 3}

This means that we cannot index it.

1. >>> myset[0]

*Traceback (most recent call last):*  
*File “<pyshell#197>”, line 1, in <module>*  
*myset[0]*  
*TypeError: ‘set’ object does not support indexing*

However, a set is mutable.[A frozen set is immutable](https://data-flair.training/blogs/python-built-in-functions/). This means we cannot change its values. This also makes it eligible to be used as a key for a dictionary.

1. >>> myset=frozenset([1,3,2,2])
2. >>> myset

frozenset({1, 2, 3})

1. >>> type(myset)

<class ‘frozenset’>

[94] **How would you generate a random number in Python?**

This kind of Python interview Questions and Answers can Prove your depth of knowledge.

To generate a random number, we import the function random() from the module random.

1. >>> from random import random
2. >>> random()

0.7931961644126482

Let’s call for help on this.

1. >>> help(random)

[Help on built-in function](https://data-flair.training/blogs/python-built-in-functions/)random:

random(…) method of random.Random instance  
random() -> x in the interval [0, 1).

This means that it will return a random number equal to or greater than 0, and less than 1.

We can also use the function randint(). It takes two arguments to indicate a range from which to return a random integer.

1. >>> from random import randint
2. >>> randint(2,7)

6

1. >>> randint(2,7)

5

1. >>> randint(2,7)

7

1. >>> randint(2,7)

6

1. >>> randint(2,7)

2

[95] **How will you capitalize the first letter of a string?**

Simply using the [method capitalize().](https://data-flair.training/blogs/python-strings/)

1. >>> 'ayushi'.capitalize()

‘Ayushi’

1. >>> type(str.capitalize)

<class ‘method\_descriptor’>

However, it will let other characters be.

1. >>> '@yushi'.capitalize()

‘@yushi’

[96]How will you check if all characters in a string are alphanumeric?

For this, we use the method isalnum().

1. >>> 'Ayushi123'.isalnum()

True

1. >>> 'Ayushi123!'.isalnum()

False

Other methods that we have include:

1. >>> '123.3'.isdigit()

False

1. >>> '123'.isnumeric()

True

1. >>> 'ayushi'.islower()

True

1. >>> 'Ayushi'.isupper()

False

1. >>> 'Ayushi'.istitle()

True

1. >>> ' '.isspace()

True

1. >>> '123F'.isdecimal()

False

[97]Explain lambda expressions. When would you use one?

When we want a function with a single expression, we can define it anonymously. A lambda expression may take input and returns a value. To define the above function as a lambda expression, we type the following code in the interpreter:

1. >>> (lambda a,b:a if a>b else b)(3,3.5)

3.5

Here, a and b are the inputs. a if a>b else b is the expression to return. The arguments are 3 and 3.5.  
It is possible to not have any inputs here.

1. >>> (lambda :print("Hi"))()

Hi

[98]What is a generator?

[Python generator](https://data-flair.training/blogs/python-generator/) produces a sequence of values to iterate on. This way, it is kind of an iterable.

We define a function that ‘yields’ values one by one, and then use a for loop to iterate on it.

1. >>> def squares(n):
2. i=1
3. while(i<=n):
4. yield i\*\*2
5. i+=1
6. >>> for i in squares(7):
7. print(i)

1  
4  
9  
16  
25  
36  
49

[99]So, what is an iterator, then?

An iterator returns one object at a time to iterate on. To create an iterator, we use the iter() function.  
odds=iter([1,3,5,7,9])

Then, we call the next() function on it every time we want an object.

1. >>> next(odds)

1

1. >>> next(odds)

3

1. >>> next(odds)

5

1. >>> next(odds)

7

1. >>> next(odds)

9  
And now, when we call it again, it raises a StopIteration exception. This is because it has reached the end of the values to iterate on.

1. >>> next(odds)

Traceback (most recent call last):  
File “<pyshell#295>”, line 1, in <module>  
next(odds)  
StopIteration

[100]Okay, we asked you about generators and iterators, and you gave us the right answers. But don’t they sound similar?

They do, but there are subtle differences:

1. For a generator, we create a function. For an iterator, we use in-built functions iter() and next().
2. For a generator, we use the keyword ‘yield’ to yield/return an object at a time.
3. A generator may have as many ‘yield’ statements as you want.
4. A generator will save the states of the local variables every time ‘yield’ will pause the loop. An iterator does not use local variables; it only needs an iterable to iterate on.
5. Using a class, you can implement your own iterator, but not a generator.
6. Generators are fast, compact, and simpler.
7. Iterators are more memory-efficient.

[100]What does the function zip() do?

One of the less common functions with beginners, zip() returns an iterator of tuples.

1. >>> list(zip(['a','b','c'],[1,2,3]))

[(‘a’, 1), (‘b’, 2), (‘c’, 3)]

Here, it pairs items from the two lists, and creates tuples with those. But it doesn’t have to be lists.

1. >>> list(zip(('a','b','c'),(1,2,3)))

[(‘a’, 1), (‘b’, 2), (‘c’, 3)]

[101]How will you find, in a string, the first word that rhymes with ‘cake’?

For our purpose, we will use the function search(), and then use group() to get the output.

1. >>> import re
2. >>> rhyme=re.search('.ake','I would make a cake, but I hate to bake')
3. >>> rhyme.group()

‘make’

And as we know, the function search() stops at the first match. Hence, we have our first rhyme to ‘cake’.

[102]How would you display a file’s contents in reversed order?

Let’s first get to the Desktop. We use the chdir() function/method form the os module for this.

1. >>> import os
2. >>> os.chdir('C:\\Users\\lifei\\Desktop')

The file we’ll use for this is Today.txt, and it has the following contents:  
OS, DBMS, DS, ADA  
HTML, CSS, jQuery, JavaScript  
Python, C++, Java  
This sem’s subjects  
Debugger  
itertools  
container

Let’s read the contents into a list, and then call reversed() on it:

1. >>> for line in reversed(list(open('Today.txt'))):
2. print(line.rstrip())

container  
itertools  
Debugger

This sem’s subjects

Python, C++, Java

HTML, CSS, jQuery, JavaScript

OS, DBMS, DS, ADA

Without the rstrip(), we would get blank lines between the output.

[103] **What is Tkinter?**

[Tkinter is a famous Python library](https://data-flair.training/blogs/python-gui-programming/) with which you can craft a GUI. It provides support for different GUI tools and widgets like buttons, labels, text boxes, radio buttons, and more. These tools and widgets have attributes like dimensions, colors, fonts, colors, and more.

You can also import the tkinter module.

1. >>> import tkinter
2. >>> top=tkinter.Tk()

This will create a new window for you:

This creates a window with the title ‘My Game’. You can position your widgets on this.

[104] How is a .pyc file different from a .py file?

While both files hold bytecode, .pyc is the compiled version of a Python file. It has platform-independent bytecode. Hence, we can execute it on any platform that supports the .pyc format. Python automatically generates it to improve performance(in terms of load time, not speed).

[105] **How do you create your own package in Python?**

We know that a package may contain sub-packages and modules. A module is nothing but Python code.

To create a package of our own, we create a [directory](https://data-flair.training/blogs/python-directory/) and create a file \_\_init\_\_.py in it. We leave it empty. Then, in that package, we create a [module](https://data-flair.training/blogs/python-modules/)(s) with whatever code we want. For a detailed explanation with pictures, refer to [Python Packages](https://data-flair.training/blogs/python-packages/).

[106]How do you calculate the length of a string?

This is simple. We call the function len() on the string we want to calculate the length of.

1. >>> len('Ayushi Sharma')

13

# [107] Class or Static Variables in Python

Class or static variables are shared by all objects. Instance or non-static variables are different for different objects (every object has a copy of it).

For example, let a Computer Science Student be represented by class CSStudent. The class may have a static variable whose value is “cse” for all objects. And class may also have non-static members like name and roll.

In [C++](https://www.geeksforgeeks.org/c-plus-plus/) and [Java](https://www.geeksforgeeks.org/java/), we can use static keyword to make a variable as class variable. The variables which don’t have preceding static keyword are instance variables. See [this](https://www.geeksforgeeks.org/static-keyword-in-java/) for Java example and [this](https://www.geeksforgeeks.org/stati/) for C++ example.

The Python approach is simple, it doesn’t require a static keyword. All variables which are assigned a value in class declaration are class variables. And variables which are assigned values inside class methods are instance variables.

|  |
| --- |
| # Python program to show that the variables with a value  # assigned in class declaration, are class variables    # Class for Computer Science Student  class CSStudent:      stream = 'cse'                  # Class Variable      def \_\_init\_\_(self,name,roll):          self.name = name            # Instance Variable          self.roll = roll            # Instance Variable    # Objects of CSStudent class  a = CSStudent('Geek', 1)  b = CSStudent('Nerd', 2)    print(a.stream)  # prints "cse"  print(b.stream)  # prints "cse"  print(a.name)    # prints "Geek"  print(b.name)    # prints "Nerd"  print(a.roll)    # prints "1"  print(b.roll)    # prints "2"    # Class variables can be accessed using class  # name also  print(CSStudent.stream) # prints "cse" |

Output:

cse

cse

Geek

Nerd

1

2

cse

# [108] class method vs static method in Python

The @classmethod decorator, is a builtin [function decorator](https://www.geeksforgeeks.org/function-decorators-in-python-set-1-introduction/) that is an expression that gets evaluated after your function is defined. The result of that evaluation shadows your function definition.  
A class method receives the class as implicit first argument, just like an instance method receives the instance  
**Syntax:**

**class C(object):**

**@classmethod**

**def fun(cls, arg1, arg2, ...):**

....

**fun:** function that needs to be converted into a class method

**returns:** a class method for function.

* A class method is a method which is bound to the class and not the object of the class.
* They have the access to the state of the class as it takes a class parameter that points to the class and not the object instance.
* It can modify a class state that would apply across all the instances of the class. For example it can modify a class variable that will be applicable to all the instances.

**Static Method**

A static method does not receive an implicit first argument.  
**Syntax:**

**class C(object):**

**@staticmethod**

**def fun(arg1, arg2, ...):**

**...**

**returns:** a static method for function fun.

* A static method is also a method which is bound to the class and not the object of the class.
* A static method can’t access or modify class state.
* It is present in a class because it makes sense for the method to be present in class.

Class method vs Static Method

* A class method takes cls as first parameter while a static method needs no specific parameters.
* A class method can access or modify class state while a static method can’t access or modify it.
* In general, static methods know nothing about class state. They are utility type methods that take some parameters and work upon those parameters. On the other hand class methods must have class as parameter.
* We use @classmethod decorator in python to create a class method and we use @staticmethod decorator to create a static method in python.

When to use what?

* We generally use class method to create factory methods. Factory methods return class object ( similar to a constructor ) for different use cases.
* We generally use static methods to create utility functions.

**How to define a class method and a static method?**

To define a class method in python, we use @classmethod decorator and to define a static method we use @staticmethod decorator.  
Let us look at an example to understand the difference between both of them. Let us say we want to create a class Person. Now, python doesn’t support method overloading like C++ or Java so we use class methods to create factory methods. In the below example we use a class method to create a person object from birth year.

As explained above we use static methods to create utility functions. In the below example we use a static method to check if a person is adult or not.

**Implementation**

|  |
| --- |
| # Python program to demonstrate  # use of class method and static method.  from datetime import date    class Person:      def \_\_init\_\_(self, name, age):          self.name = name          self.age = age        # a class method to create a Person object by birth year.      @classmethod      def fromBirthYear(cls, name, year):          return cls(name, date.today().year - year)        # a static method to check if a Person is adult or not.      @staticmethod      def isAdult(age):          return age > 18    person1 = Person('mayank', 21)  person2 = Person.fromBirthYear('mayank', 1996)    print person1.age  print person2.age    # print the result  print Person.isAdult(22) |

Output

21

21

True

# [109] Decorators in Python

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.

|  |
| --- |
| # defining a decorator  def hello\_decorator(func):        # inner1 is a Wrapper function in      # which the argument is called        # inner function can access the outer local      # functions like in this case "func"      def inner1():          print("Hello, this is before function execution")            # calling the actual function now          # inside the wrapper function.          func()            print("This is after function execution")        return inner1      # defining a function, to be called inside wrapper  def function\_to\_be\_used():      print("This is inside the function !!")      # passing 'function\_to\_be\_used' inside the  # decorator to control its behavior  function\_to\_be\_used = hello\_decorator(function\_to\_be\_used)      # calling the function  function\_to\_be\_used() |

Output:

Hello, this is before function execution

This is inside the function !!

This is after function execution

# [110] Transpose a matrix in Single line in Python

[Transpose](https://www.geeksforgeeks.org/inplace-m-x-n-size-matrix-transpose/) of a matrix is a task we all can perform very easily in python (Using a nested loop). But there are some interesting ways to do the same in a single line.  
In Python, we can implement a matrix as nested list (list inside a list). Each element is treated as a row of the matrix. For example m = [[1, 2], [4, 5], [3, 6]] represents a matrix of 3 rows and 2 columns.  
First element of the list – **m[0]** and element in first row, first column – **m[0][0]**.

1. **Using Nested List Comprehension:**Nested list comprehension are used to iterate through each element in the matrix.In the given example ,we iterate through each element of matrix (m) in column major manner and assign the result to rez matrix which is the transpose of m.

m = [[1,2],[3,4],[5,6]]

for row in m :

    print(row)

rez = [[m[j][i] for j in range(len(m))] for i in range(len(m[0]))]

print("\n")

for row in rez:

    print(row)

**Output:**

[1, 2]

[3, 4]

[5, 6]

[1, 3, 5]

[2, 4, 6]

**Using zip:**Zip returns an iterator of tuples, where the i-th tuple contains the i-th element from each of the argument sequences or iterables. In this example we unzip our array using \* and then zip it to get the transpose.

|  |
| --- |
| matrix=[(1,2,3),(4,5,6),(7,8,9),(10,11,12)]  for row in matrix:      print(row)  print("\n")  t\_matrix = zip(\*matrix)  for row in t\_matrix:      print(row) |

Output:

(1, 2, 3)

(4, 5, 6)

(7, 8, 9)

(10, 11, 12)

(1, 4, 7, 10)

(2, 5, 8, 11)

(3, 6, 9, 12)

**Using numpy:**NumPy is a general-purpose array-processing package designed to efficiently manipulate large multi-dimensional arrays. The transpose method returns a transposed view of the passed multi-dimensional matrix.

# You need to install numpy in order to import it

# Numpy transpose returns similar result when

# applied on 1D matrix

import numpy

matrix=[[1,2,3],[4,5,6]]

print(matrix)

print("\n")

print(numpy.transpose(matrix))

# [111] Global keyword in Python

Global keyword is a keyword that allows a user to modify a variable outside of the current scope. It is used to create [global variables](https://www.geeksforgeeks.org/global-local-variables-python/) from a non-global scope i.e inside a function. Global keyword is used inside a function only when we want to do assignments or when we want to change a variable. Global is not needed for printing and accessing.

**Rules of global keyword:**

* If a variable is assigned a value anywhere within the function’s body, it’s assumed to be a local unless explicitly declared as global.
* Variables that are only referenced inside a function are implicitly global.
* We Use global keyword to use a global variable inside a function.
* There is no need to use global keyword outside a function.

**Use of global keyword:**  
To access a global variable inside a function there is no need to use global keyword.  
**Example 1:**

|  |
| --- |
| # Python program showing no need to  # use global keyword for accessing  # a global value    # global variable  a = 15  b = 10    # function to perform addition  def add():      c = a + b      print(c)    # calling a function  add() |

Output:

25

If we need to assign a new value to a global variable then we can do that by declaring the variable as global.  
**Code 2:** Without global keyword

|  |
| --- |
| # Python program showing to modify  # a global value without using global  # keyword    a = 15    # function to change a global value  def change():        # increment value of a by 5      a = a + 5      print(a)    change() |

Output:

UnboundLocalError: local variable 'a' referenced before assignment

This output is an error because we are trying to assign a value to a variable in an outer scope. This can be done with the use of global variable.  
**Code 3 :** With global keyword

|  |
| --- |
| # Python program to modify a global  # value inside a function    x = 15  def change():        # using a global keyword      global x        # increment value of a by 5      x = x + 5      print("Value of x inside a function :", x)  change()  print("Value of x outside a function :", x) |

Output:

Value of x inside a function : 20

Value of x outside a function : 20

**Global in Nested functions**  
In order to use global inside a nested functions, we have to declare a variable with global keyword inside a nested function

|  |
| --- |
| # Python program showing a use of  # global in nested function    def add():       x = 15         def change():           global x           x = 20       print("Before making changing: ", x)       print("Making change")       change()       print("After making change: ", x)    add()  print("value of x",x) |

Output:

Before making changing: 15

Making change

After making change: 15

value of x 20

# [112] try and except in Python

**try()** is used in **Error and Exception Handling**  
There are two kinds of errors :

* **Syntax Error** : Also known as Parsing Errors, most basic. Arise when the Python parser is unable to understand a line of code.
* **Exception** : Errors which are detected during execution. eg – ZeroDivisionError.

**List of Exception Errors :**

* **IOError :**if file can’t be opened
* **KeyboardInterrupt :**when an unrequired key is pressed by the user
* **ValueError :**when built-in function receives a wrong argument
* **EOFError :**if End-Of-File is hit without reading any data
* **ImportError :**if it is unable to find the module

Now, here comes the task to handle these errors within our code in Python. So here we need try-except statements.

Basic Syntax :

try:

// Code

except:

// Code

How try() works?

* First try clause is executed i.e. the code between try and except clause.
* If there is no exception, then only try clause will run, except clause is finished.
* If any exception occured, try clause will be skipped and except clause will run.
* If any exception occurs, but the except clause within the code doesn’t handle it, it is passed on to the outer try statements. If the exception left unhandled, then the execution stops.
* A try statement can have more than one except clause

Code 1 : No exception, so try clause will run.

|  |
| --- |
| # Python code to illustrate  # working of try()  def divide(x, y):      try:          # Floor Division : Gives only Fractional Part as Answer          result = x // y          print("Yeah ! Your answer is :", result)      except ZeroDivisionError:          print("Sorry ! You are dividing by zero ")    # Look at parameters and note the working of Program  divide(3, 2) |

Output :

('Yeah ! Your answer is :', 1)

**Code 1 :**There is an exception so only **except** clause will run.

|  |
| --- |
| # Python code to illustrate  # working of try()  def divide(x, y):      try:          # Floor Division : Gives only Fractional Part as Answer          result = x // y          print("Yeah ! Your answer is :", result)      except ZeroDivisionError:          print("Sorry ! You are dividing by zero ")    # Look at parameters and note the working of Program  divide(3, 0) |

Output :

Sorry ! You are dividing by zero

# [113] Python | Set 5 (Exception Handling)

|  |
| --- |
| # Python program to handle simple runtime error    a = [1, 2, 3]  try:      print "Second element = %d" %(a[1])        # Throws error since there are only 3 elements in array      print "Fourth element = %d" %(a[3])    except IndexError:      print "An error occurred" |

Output:

Second element = 2

An error occurred

A try statement can have more than one except clause, to specify handlers for different exceptions. Please note that at most one handler will be executed.

|  |
| --- |
| # Program to handle multiple errors with one except statement  try :      a = 3      if a < 4 :            # throws ZeroDivisionError for a = 3          b = a/(a-3)        # throws NameError if a >= 4      print "Value of b = ", b    # note that braces () are necessary here for multiple exceptions  except(ZeroDivisionError, NameError):      print "\nError Occurred and Handled" |

Output:

Error Occurred and Handled

If you change the value of ‘a’ to greater than or equal to 4, the the output will be

Value of b =

Error Occurred and Handled

The output above is so because as soon as python tries to access the value of b, NameError occurs.  
   
Else Clause:  
In python, you can also use else clause on try-except block which must be present after all the except clauses. The code enters the else block only if the try clause does not raise an exception.

|  |
| --- |
| # Program to depict else clause with try-except    # Function which returns a/b  def AbyB(a , b):      try:          c = ((a+b) / (a-b))      except ZeroDivisionError:          print "a/b result in 0"      else:          print c    # Driver program to test above function  AbyB(2.0, 3.0)  AbyB(3.0, 3.0) |

The output for above program will be :

-5.0

a/b result in 0

Raising Exception:  
The raise statement allows the programmer to force a specific exception to occur. The sole argument in raise indicates the exception to be raised. This must be either an exception instance or an exception class (a class that derives from Exception).

|  |
| --- |
| # Program to depict Raising Exception    try:      raise NameError("Hi there")  # Raise Error  except NameError:      print "An exception"      raise  # To determine whether the exception was raised or not |

The output of the above code will simply line printed as “An exception” but a Runtime error will also occur in the last due to raise statement in the last line. So, the output on your command line will look like

Traceback (most recent call last):

File "003dff3d748c75816b7f849be98b91b8.py", line 4, in

raise NameError("Hi there") # Raise Error

NameError: Hi there

# [114] Python | Check if all the values in a list are less than a given value

Given a list, write a Python program to check if all the values in a list are less than the given value.

**Examples:**

**Input :** list = [11, 22, 33, 44, 55]

value = 22

**Output :** No

**Input :** list = [11, 22, 33, 44, 55]

value = 65

**Output :** Yes

**Method #1:** Traversing the list  
Compare each element by iterating through the list and check if all the elements in the given list are less than the given value or not.

|  |
| --- |
| # Python program to check if all values  # in the list are less than given value    # Function to check the value  def CheckForLess(list1, val):        # traverse in the list      for x in list1:            # compare with all the          # values with value          if val <= x:              return False      return True    # Driver code  list1 = [11, 22, 33, 44, 55]  val = 65  if (CheckForLess(list1, val)): print("Yes")  else: print("No") |

Output:

Yes

**Method #2:** Using all() function  
Using all() function we can check if all values are less than any given value in a single line. It returns true if the given condition inside the all() function is true for all values, else it returns false.

|  |
| --- |
| # Python program to check if all values  # in the list are less than given value    # Function to check the value  def CheckForLess(list1, val):      return(all(x < val for x in list1))    # Driver code  list1 = [11, 22, 33, 44, 55]  val = 65  if (CheckForLess(list1, val)): print("Yes")  else: print("No") |

Output:

Yes

[115]

[116]

[117]

[118]

[119]

[120]

# Python Itertools Tutorial

# [1] The Python itertools module has functions for creating iterators for efficient looping. While some iterators are infinite, some terminate on the shortest input sequence. Yet, some are combinatoric. Let’s first discuss infinite iterators.

Basic Programs in Python

[1]

# Python Program for factorial of a number

Factorial of a non-negative integer, is multiplication of all integers smaller than or equal to n. For example factorial of 6 is 6\*5\*4\*3\*2\*1 which is 720.

# Python 3 program to find

# factorial of given number

def factorial(n):

    # single line to find factorial

    return 1 if (n==1 or n==0) else n \* factorial(n - 1);

# Driver Code

num = 5;

print("Factorial of",num,"is",

factorial(num))

# [2] Python Program for Fibonacci numbers

The Fibonacci numbers are the numbers in the following integer sequence.

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ……..

In mathematical terms, the sequence Fn of Fibonacci numbers is defined by the recurrence relation

Fn = Fn-1 + Fn-2

with seed values

F0 = 0 and F1 = 1.

**Method 1 ( Use recursion ) :**

|  |
| --- |
| # Function for nth Fibonacci number    def Fibonacci(n):      if n<0:          print("Incorrect input")      # First Fibonacci number is 0      elif n==1:          return 0      # Second Fibonacci number is 1      elif n==2:          return 1      else:          return Fibonacci(n-1)+Fibonacci(n-2)    # Driver Program |

Output:

21

**Method 2 ( Use Dynamic Programming ) :**

|  |
| --- |
| # Function for nth fibonacci number - Dynamic Programing  # Taking 1st two fibonacci nubers as 0 and 1    FibArray = [0,1]    def fibonacci(n):      if n<0:          print("Incorrect input")      elif n<=len(FibArray):          return FibArray[n-1]      else:          temp\_fib = fibonacci(n-1)+fibonacci(n-2)          FibArray.append(temp\_fib)          return temp\_fib    # Driver Program    print(fibonacci(9))    #This code is contributed by Saket Modi |

Output:

21

**Method 3 ( Space Optimized):**

|  |
| --- |
| # Function for nth fibonacci number - Space Optimisataion  # Taking 1st two fibonacci numbers as 0 and 1    def fibonacci(n):      a = 0      b = 1      if n < 0:          print("Incorrect input")      elif n == 0:          return a      elif n == 1:          return b      else:          for i in range(2,n):              c = a + b              a = b              b = c          return b    # Driver Program    print(fibonacci(9))    #This code is contributed by Saket Modi |

Output:

21

[3] Python Program for cube sum of first n natural numbers

Print the sum of series 13 + 23 + 33 + 43 + …….+ n3 till n-th term.

Examples:

Input : n = 5

Output : 225

13 + 23 + 33 + 43 + 53 = 225

Input : n = 7

Output : 784

13 + 23 + 33 + 43 + 53 +

63 + 73 = 784

|  |
| --- |
| # Simple Python program to find sum of series  # with cubes of first n natural numbers    # Returns the sum of series  def sumOfSeries(n):      sum = 0      for i in range(1, n+1):          sum +=i\*i\*i        return sum      # Driver Function  n = 5  print(sumOfSeries(n))    # Code Contributed by Mohit Gupta\_OMG <(0\_o)> |

Output :

225

[4] Python program to check if a string is palindrome or not

Given a string, write a python function to check if it is palindrome or not. A string is said to be palindrome if reverse of the string is same as string. For example, “radar” is palindrome, but “radix” is not palindrome.

Examples:

Input : malayalam

Output : Yes

Input : geeks

Output : No

|  |
| --- |
| # function which return reverse of a string  def reverse(s):      return s[::-1]    def isPalindrome(s):      # Calling reverse function      rev = reverse(s)        # Checking if both string are equal or not      if (s == rev):          return True      return False      # Driver code  s = "malayalam"  ans = isPalindrome(s)    if ans == 1:      print("Yes")  else:      print("No") |

Output :

Yes

|  |
| --- |
| # function to check string is  # palindrome or not  def isPalindrome(s):        # Using predefined function to      # reverse to string print(s)      rev = ''.join(reversed(s))        # Checking if both string are      # equal or not      if (s == rev):          return True      return False    # main function  s = "malayalam"  ans = isPalindrome(s)    if (ans):      print("Yes")  else:      print("No") |

Output:

Yes

[5] Python program to check if given string is pangram

Given a string, write a Python program to check if that string is Pangram or not. A pangram is a sentence containing every letter in the English Alphabet.

Examples:

Input : The quick brown fox jumps over the lazy dog

Output : Yes

Input : abcdefgxyz

Output : No

We have already discussed the naive approach of pangram checking in [this article](https://www.geeksforgeeks.org/pangram-checking/). Now, let’s discuss the Pythonic approaches to do the same.

Approach #1 : Pythonic Naive  
This method uses a loop to check if each character of the string belongs to the alphabet set or not.

|  |
| --- |
| # Python3 program to  # Check if the string is pangram  import string    def ispangram(str):      alphabet = "abcdefghijklmnopqrstuvwxyz"      for char in alphabet:          if char not in str.lower():              return False        return True    # Driver code  string = 'the quick brown fox jumps over the lazy dog'  if(ispangram(string) == True):      print("Yes")  else:      print("No") |

Output:

Yes

Approach #2 : Using Python Set  
Convert the given string into set and then check if the alphabet set is greater than or equal to it or not. If the string set is greater or equal, print ‘Yes’ otherwise ‘No’.

|  |
| --- |
| # Python3 program to  # Check if the string is pangram  import string    alphabet = set(string.ascii\_lowercase)    def ispangram(string):      return set(string.lower()) >= alphabet    # Driver code  string = "The quick brown fox jumps over the lazy dog"  if(ispangram(string) == True):      print("Yes")  else:      print("No") |

Output:

Yes

Approach #3 : Alternative to set method  
This is another method that uses Python set to find if the string is Pangram or not. We make set of lowercase alphabets and the given string. If set of given string is subtracted from the set of alphabets, we get to know whether the string is pangram or not.

|  |
| --- |
| # Python3 program to  # Check if the string is pangram  import string    alphabet = set(string.ascii\_lowercase)    def ispangram(str):       return not set(alphabet) - set(str)    # Driver code  string = 'the quick brown fox jumps over the lazy dog'  if(ispangram(string) == True):      print("Yes")  else:      print("No") |

Output:

Yes

[6]Infinite Fibonacci series using generator

def fib():

a,b = 0,1

while True:

yield a

a,b = b, a+b

for val in fib():

print(val)

[7]

[8]

[9]

[10]

[11]

[12]

[13]

[14]

[15]

[16]

[17]