**Python Functions**

In Python, function is a group of related statements that perform a specific task.Functions help break our program into smaller and modular chunks. As our program grows larger and larger, functions make it more organized and manageable.

Furthermore, it avoids repetition and makes code reusable.

Syntax of Function

def function\_name(parameters):

"""docstring"""

statement(s)

Above shown is a function definition which consists of following components.

* Keyword def marks the start of function header.
* A function name to uniquely identify it. Function naming follows the same rules of writing identifiers in Python.
* Parameters (arguments) through which we pass values to a function. They are optional.
* A colon (:) to mark the end of function header.
* Optional documentation string (docstring) to describe what the function does.
* One or more valid python statements that make up the function body. Statements must have same indentation level (usually 4 spaces).
* An optional return statement to return a value from the function.

Example of a function

def greet(name):

"""This function greets to

the person passed in as

parameter"""

print("Hello, " + name + ". Good morning!")

Function Call

Once we have defined a function, we can call it from another function, program or even the Python prompt. To call a function we simply type the function name with appropriate parameters.

>>> greet('Paul')

Hello, Paul. Good morning!

Note: Try running the above code into the Python shell to see the output.

Docstring

The first string after the function header is called the docstring and is short for documentation string. It is used to explain in brief, what a function does.

Although optional, documentation is a good programming practice. Unless you can remember what you had for dinner last week, always document your code.

In the above example, we have a docstring immediately below the function header. We generally use triple quotes so that docstring can extend up to multiple lines. This string is available to us as \_\_doc\_\_ attribute of the function.

For example:

Try running the following into the Python shell to see the output.

>>> print(greet.\_\_doc\_\_)

This function greets to

the person passed into the

name parameter

The return statement

The return statement is used to exit a function and go back to the place from where it was called.

Syntax of return

return [expression\_list]

This statement can contain expression which gets evaluated and the value is returned. If there is no expression in the statement or the return statement itself is not present inside a function, then the function will return the None object.

For example:

>>> print(greet("May"))

Hello, May. Good morning!

None

Here, None is the returned value.

Example of return

def absolute\_value(num):

"""This function returns the absolute

value of the entered number"""

if num >= 0:

return num

else:

return -num

# Output: 2

print(absolute\_value(2))

# Output: 4

print(absolute\_value(-4))

How Function works in Python?

How function works in Python?

Scope and Lifetime of variables

Scope of a variable is the portion of a program where the variable is recognized. Parameters and variables defined inside a function is not visible from outside. Hence, they have a local scope.

Lifetime of a variable is the period throughout which the variable exits in the memory. The lifetime of variables inside a function is as long as the function executes.

They are destroyed once we return from the function. Hence, a function does not remember the value of a variable from its previous calls.

Here is an example to illustrate the scope of a variable inside a function.

def my\_func():

x = 10

print("Value inside function:",x)

x = 20

my\_func()

print("Value outside function:",x)

Output

Value inside function: 10

Value outside function: 20

Here, we can see that the value of x is 20 initially. Even though the function my\_func() changed the value of x to 10, it did not effect the value outside the function.

This is because the variable x inside the function is different (local to the function) from the one outside. Although they have same names, they are two different variables with different scope.

On the other hand, variables outside of the function are visible from inside. They have a global scope.

We can read these values from inside the function but cannot change (write) them. In order to modify the value of variables outside the function, they must be declared as global variables using the keyword global.

Types of Functions

Basically, we can divide functions into the following two types:

Built-in functions - Functions that are built into Python.

User-defined functions - Functions defined by the users themselves.

Builtin Functions:

dir(\_\_builtins\_\_)

User Defined Functions:

In user-defined function topic, we learned about defining a function and calling it. Otherwise, the function call will result into an error. Here is an example.

def greet(name,msg):

"""This function greets to

the person with the provided message"""

print("Hello",name + ', ' + msg)

greet("Nikhil","Good morning!")

Output

Hello Nikhil, Good morning!

Here, the function greet() has two parameters.

Since, we have called this function with two arguments, it runs smoothly and we do not get any error.

If we call it with different number of arguments, the interpreter will complain. Below is a call to this function with one and no arguments along with their respective error messages.

>>> greet("Monica") # only one argument

TypeError: greet() missing 1 required positional argument: 'msg'

>>> greet() # no arguments

TypeError: greet() missing 2 required positional arguments: 'name' and 'msg'

Variable Function Arguments

Up until now functions had fixed number of arguments. In Python there are other ways to define a function which can take variable number of arguments.

Three different forms of this type are described below.

**Python Default Arguments:**

Function arguments can have default values in Python

We can provide a default value to an argument by using the assignment operator (=). Here is an example.

def greet(name, msg = "Good morning!"):

"""

This function greets to

the person with the

provided message.

If message is not provided,

it defaults to "Good

morning!"

"""

print("Hello",name + ', ' + msg)

greet("Kate")

greet("Bruce","How do you do?")

In this function, the parameter name does not have a default value and is required (mandatory) during a call.

On the other hand, the parameter msg has a default value of "Good morning!". So, it is optional during a call. If a value is provided, it will overwrite the default value.

Any number of arguments in a function can have a default value. But once we have a default argument, all the arguments to its right must also have default values.

This means to say, non-default arguments cannot follow default arguments. For example, if we had defined the function header above as:

def greet(msg = "Good morning!", name):

We would get an error as:

SyntaxError: non-default argument follows default argument

**Python Keyword Arguments**

When we call a function with some values, these values get assigned to the arguments according to their position.

For example, in the above function greet(), when we called it as greet("Bruce","How do you do?"), the value "Bruce" gets assigned to the argument name and similarly "How do you do?" to msg.

Python allows functions to be called using keyword arguments. When we call functions in this way, the order (position) of the arguments can be changed. Following calls to the above function are all valid and produce the same result.

>>> # 2 keyword arguments

>>> greet(name = "Bruce",msg = "How do you do?")

>>> # 2 keyword arguments (out of order)

>>> greet(msg = "How do you do?",name = "Bruce")

>>> # 1 positional, 1 keyword argument

>>> greet("Bruce",msg = "How do you do?")

As we can see, we can mix positional arguments with keyword arguments during a function call. But we must keep in mind that keyword arguments must follow positional arguments.

Having a positional argument after keyword arguments will result into errors. For example the function call as follows:

greet(name="Bruce","How do you do?")

Will result into error as:

SyntaxError: non-keyword arg after keyword arg

**Python Arbitrary Arguments**

Sometimes, we do not know in advance the number of arguments that will be passed into a function.Python allows us to handle this kind of situation through function calls with arbitrary number of arguments.

In the function definition we use an asterisk (\*) before the parameter name to denote this kind of argument. Here is an example.

def greet(\*names):

"""This function greets all

the person in the names tuple."""

# names is a tuple with arguments

for name in names:

print("Hello",name)

greet("Monica","Luke","Steve","John")

Output

Hello Monica

Hello Luke

Hello Steve

Hello John

Here, we have called the function with multiple arguments. These arguments get wrapped up into a tuple before being passed into the function. Inside the function, we use a for loop to retrieve all the arguments back.

**Python Recursion**

We will learn to create a recursive function; a function that calls itself.

Working of recursive function in Python

Recursion is the process of defining something in terms of itself.

A physical world example would be to place two parallel mirrors facing each other. Any object in between them would be reflected recursively.

**Python Recursive Function**

We know that in Python, a function can call other functions. It is even possible for the function to call itself. These type of construct are termed as recursive functions.

Following is an example of recursive function to find the factorial of an integer.

Factorial of a number is the product of all the integers from 1 to that number. For example, the factorial of 6 (denoted as 6!) is 1\*2\*3\*4\*5\*6 = 720.

Example of recursive function

# An example of a recursive function to

# find the factorial of a number

def calc\_factorial(x):

"""This is a recursive function

to find the factorial of an integer"""

if x == 1:

return 1

else:

return (x \* calc\_factorial(x-1))

num = 4

print("The factorial of", num, "is", calc\_factorial(num))

In the above example, calc\_factorial() is a recursive functions as it calls itself.

When we call this function with a positive integer, it will recursively call itself by decreasing the number.

Each function call multiples the number with the factorial of number 1 until the number is equal to one. This recursive call can be explained in the following steps.

calc\_factorial(4) # 1st call with 4

4 \* calc\_factorial(3) # 2nd call with 3

4 \* 3 \* calc\_factorial(2) # 3rd call with 2

4 \* 3 \* 2 \* calc\_factorial(1) # 4th call with 1

4 \* 3 \* 2 \* 1 # return from 4th call as number=1

4 \* 3 \* 2 # return from 3rd call

4 \* 6 # return from 2nd call

24 # return from 1st call

Our recursion ends when the number reduces to 1. This is called the base condition.

Every recursive function must have a base condition that stops the recursion or else the function calls itself infinitely.

**Advantages of recursion**

Recursive functions make the code look clean and elegant.

A complex task can be broken down into simpler sub-problems using recursion.

Sequence generation is easier with recursion than using some nested iteration.

**Disadvantages of recursion**

Sometimes the logic behind recursion is hard to follow through.

Recursive calls are expensive (inefficient) as they take up a lot of memory and time.

Recursive functions are hard to debug.

**Python Anonymous/Lambda Function**

In this article, you'll learn about the anonymous function, also known as lambda functions. You'll learn what is it, its syntax and how to use it (with examples).

Python anonymous lambda function

In Python, anonymous function is a function that is defined without a name.

While normal functions are defined using the def keyword, in Python anonymous functions are defined using the lambda keyword.

Hence, anonymous functions are also called lambda functions.

**Lambda Functions**

A lambda function has the following syntax.

**Syntax of Lambda Function**

lambda arguments: expression

Lambda functions can have any number of arguments but only one expression. The expression is evaluated and returned. Lambda functions can be used wherever function objects are required.

Example of Lambda Function

Here is an example of lambda function that doubles the input value.

# Program to show the use of lambda functions

double = lambda x: x \* 2

# Output: 10

print(double(5))

In the above program, lambda x: x \* 2 is the lambda function. Here x is the argument and x \* 2 is the expression that gets evaluated and returned.

This function has no name. It returns a function object which is assigned to the identifier double. We can now call it as a normal function. The statement

double = lambda x: x \* 2

is nearly the same as

def double(x):

return x \* 2

**Use of Lambda Function**

We use lambda functions when we require a nameless function for a short period of time.

In Python, we generally use it as an argument to a higher-order function (a function that takes in other functions as arguments). Lambda functions are used along with built-in functions like filter(), map() etc.

**Example use with filter()**

The filter() function in Python takes in a function and a list as arguments.

The function is called with all the items in the list and a new list is returned which contains items for which the function evaluats to True.

Here is an example use of filter() function to filter out only even numbers from a list.

script.py

# Program to filter out only the even items from a list

my\_list = [1, 5, 4, 6, 8, 11, 3, 12]

new\_list = list(filter(lambda x: (x%2 == 0) , my\_list))

# Output: [4, 6, 8, 12]

print(new\_list)

**Example use with map()**

The map() function in Python takes in a function and a list.

The function is called with all the items in the list and a new list is returned which contains items returned by that function for each item.

Here is an example use of map() function to double all the items in a list.

# Program to double each item in a list using map()

my\_list = [1, 5, 4, 6, 8, 11, 3, 12]

new\_list = list(map(lambda x: x \* 2 , my\_list))

# Output: [2, 10, 8, 12, 16, 22, 6, 24]

print(new\_list)

**Python Modules:**

In this article, you will learn to create and import custom modules in Python. Also, you will find different techniques to import and use custom and built-in modules in Python.

Python Modules

Modules refer to a file containing Python statements and definitions.

A file containing Python code, for e.g.: example.py, is called a module and its module name would be example.

We use modules to break down large programs into small manageable and organized files. Furthermore, modules provide reusability of code.

We can define our most used functions in a module and import it, instead of copying their definitions into different programs.

Let us create a module. Type the following and save it as example.py.

# Python Module example

def add(a, b):

"""This program adds two

numbers and return the result"""

result = a + b

return result

Here, we have defined a function add() inside a module named example. The function takes in two numbers and returns their sum.

How to import modules in Python?

We can import the definitions inside a module to another module or the interactive interpreter in Python.

We use the import keyword to do this. To import our previously defined module example we type the following in the Python prompt.

>>> import example

This does not enter the names of the functions defined in example directly in the current symbol table. It only enters the module name example there.

Using the module name we can access the function using dot (.) operation. For example:

>>> example.add(4,5.5)

9.5

Python has a ton of standard modules available.

You can check out the full list of Python standard modules and what they are for. These files are in the Lib directory inside the location where you installed Python.

Standard modules can be imported the same way as we import our user-defined modules.

There are various ways to import modules. They are listed as follows.

**Python import statement**

We can import a module using import statement and access the definitions inside it using the dot operator as described above. Here is an example.

# import statement example

# to import standard module math

import math

print("The value of pi is", math.pi)

When you run the program, the output will be:

The value of pi is 3.141592653589793

Import with renaming

We can import a module by renaming it as follows.

# import module by renaming it

import math as m

print("The value of pi is", m.pi)

We have renamed the math module as m. This can save us typing time in some cases.

Note that the name math is not recognized in our scope. Hence, math.pi is invalid, m.pi is the correct implementation.

**Python from...import statement**

We can import specific names from a module without importing the module as a whole. Here is an example.

# import only pi from math module

from math import pi

print("The value of pi is", pi)

We imported only the attribute pi from the module.

In such case we don't use the dot operator. We could have imported multiple attributes as follows.

>>> from math import pi, e

>>> pi

3.141592653589793

>>> e

2.718281828459045

Import all names

We can import all names(definitions) from a module using the following construct.

# import all names from the standard module math

from math import \*

print("The value of pi is", pi)

We imported all the definitions from the math module. This makes all names except those beginning with an underscore, visible in our scope.

Importing everything with the asterisk (\*) symbol is not a good programming practice. This can lead to duplicate definitions for an identifier. It also hampers the readability of our code.

**Python Module Search Path**

While importing a module, Python looks at several places. Interpreter first looks for a built-in module then (if not found) into a list of directories defined in sys.path. The search is in this order.

The current directory.

PYTHONPATH (an environment variable with a list of directory).

The installation-dependent default directory.

>>> import sys

>>> sys.path

['',

'C:\\Python33\\Lib\\idlelib',

'C:\\Windows\\system32\\python33.zip',

'C:\\Python33\\DLLs',

'C:\\Python33\\lib',

'C:\\Python33',

'C:\\Python33\\lib\\site-packages']

We can add modify this list to add our own path.

**Reloading a module**

The Python interpreter imports a module only once during a session. This makes things more efficient. Here is an example to show how this works.

Suppose we have the following code in a module named my\_module.

# This module shows the effect of

# multiple imports and reload

print("This code got executed")

Now we see the effect of multiple imports.

>>> import my\_module

This code got executed

>>> import my\_module

>>> import my\_module

We can see that our code got executed only once. This goes to say that our module was imported only once.

Now if our module changed during the course of the program, we would have to reload it.One way to do this is to restart the interpreter. But this does not help much.

Python provides a neat way of doing this. We can use the reload() function inside the imp module to reload a module. This is how its done.

>>> import imp

>>> import my\_module

This code got executed

>>> import my\_module

>>> imp.reload(my\_module)

This code got executed

<module 'my\_module' from '.\\my\_module.py'>

The dir() built-in function

We can use the dir() function to find out names that are defined inside a module.

For example, we have defined a function add() in the module example that we had in the beginning.

>>> dir(example)

['\_\_builtins\_\_',

'\_\_cached\_\_',

'\_\_doc\_\_',

'\_\_file\_\_',

'\_\_initializing\_\_',

'\_\_loader\_\_',

'\_\_name\_\_',

'\_\_package\_\_',

'add']

Here, we can see a sorted list of names (along with add). All other names that begin with an underscore are default Python attributes associated with the module (we did not define them ourself).

For example, the \_\_name\_\_ attribute contains the name of the module.

>>> import example

>>> example.\_\_name\_\_

'example'

All the names defined in our current namespace can be found out using the dir() function without any arguments.

>>> a = 1

>>> b = "hello"

>>> import math

>>> dir()

['\_\_builtins\_\_', '\_\_doc\_\_', '\_\_name\_\_', 'a', 'b', 'math', 'pyscripter']

**Python Package**

In this article, you'll learn to divide your code base into clean, efficient modules using Python packages. Also, you'll learn to import and use your own or third party packagesin your Python program.

Python packages

We don't usually store all of our files in our computer in the same location. We use a well-organized hierarchy of directories for easier access.

Similar files are kept in the same directory, for example, we may keep all the songs in the "music" directory. Analogous to this, Python has packages for directories and modules for files.

As our application program grows larger in size with a lot of modules, we place similar modules in one package and different modules in different packages. This makes a project (program) easy to manage and conceptually clear.

Similar, as a directory can contain sub-directories and files, a Python package can have sub-packages and modules.

A directory must contain a file named \_\_init\_\_.py in order for Python to consider it as a package. This file can be left empty but we generally place the initialization code for that package in this file.

Here is an example. Suppose we are developing a game, one possible organization of packages and modules could be as shown in the figure below.

Package Module Structure in Python Programming

Importing module from a package

We can import modules from packages using the dot (.) operator.

For example, if want to import the start module in the above example, it is done as follows.

import Game.Level.start

Now if this module contains a function named select\_difficulty(), we must use the full name to reference it.

Game.Level.start.select\_difficulty(2)

If this construct seems lengthy, we can import the module without the package prefix as follows.

from Game.Level import start

We can now call the function simply as follows.

start.select\_difficulty(2)

Yet another way of importing just the required function (or class or variable) form a module within a package would be as follows.

from Game.Level.start import select\_difficulty

Now we can directly call this function.

select\_difficulty(2)

Although easier, this method is not recommended. Using the full namespace avoids confusion and prevents two same identifier names from colliding.

While importing packages, Python looks in the list of directories defined in sys.path, similar as for module search path.

**Python File I/O**

In this article, you'll learn about Python file operations. More specifically, opening a file, reading from it, writing into it, closing it and various file methods you should be aware of.

Python file IO

File is a named location on disk to store related information. It is used to permanently store data in a non-volatile memory (e.g. hard disk).

Since, random access memory (RAM) is volatile which loses its data when computer is turned off, we use files for future use of the data.

When we want to read from or write to a file we need to open it first. When we are done, it needs to be closed, so that resources that are tied with the file are freed.

Hence, in Python, a file operation takes place in the following order.

**Open a file**

**Read or write (perform operation)**

**Close the file**

How to open a file?

Python has a built-in function open() to open a file. This function returns a file object, also called a handle, as it is used to read or modify the file accordingly.

>>> f = open("test.txt") # open file in current directory

>>> f = open("C:/Python33/README.txt") # specifying full path

We can specify the mode while opening a file. In mode, we specify whether we want to read 'r', write 'w' or append 'a' to the file. We also specify if we want to open the file in text mode or binary mode.

The default is reading in text mode. In this mode, we get strings when reading from the file.

On the other hand, binary mode returns bytes and this is the mode to be used when dealing with non-text files like image or exe files.

**Python File Modes**

Mode Description

'r' Open a file for reading. (default)

'w' Open a file for writing. Creates a new file if it does not exist or truncates the file if it exists.

'x' Open a file for exclusive creation. If the file already exists, the operation fails.

'a' Open for appending at the end of the file without truncating it. Creates a new file if it does not exist.

't' Open in text mode. (default)

'b' Open in binary mode.

'+' Open a file for updating (reading and writing)

f = open("test.txt") # equivalent to 'r' or 'rt'

f = open("test.txt",'w') # write in text mode

f = open("img.bmp",'r+b') # read and write in binary mode

Unlike other languages, the character 'a' does not imply the number 97 until it is encoded using ASCII (or other equivalent encodings).

Moreover, the default encoding is platform dependent. In windows, it is 'cp1252' but 'utf-8' in Linux.

So, we must not also rely on the default encoding or else our code will behave differently in different platforms.

Hence, when working with files in text mode, it is highly recommended to specify the encoding type.

f = open("test.txt",mode = 'r',encoding = 'utf-8')

How to close a file Using Python?

When we are done with operations to the file, we need to properly close the file.

Closing a file will free up the resources that were tied with the file and is done using Python close() method.

Python has a garbage collector to clean up unreferenced objects but, we must not rely on it to close the file.

f = open("test.txt",encoding = 'utf-8')

# perform file operations

f.close()

This method is not entirely safe. If an exception occurs when we are performing some operation with the file, the code exits without closing the file.

A safer way is to use a try...finally block.

try:

f = open("test.txt",encoding = 'utf-8')

# perform file operations

finally:

f.close()

This way, we are guaranteed that the file is properly closed even if an exception is raised, causing program flow to stop.

The best way to do this is using the with statement. This ensures that the file is closed when the block inside with is exited.

We don't need to explicitly call the close() method. It is done internally.

with open("test.txt",encoding = 'utf-8') as f:

# perform file operations

**How to write to File Using Python?**

In order to write into a file in Python, we need to open it in write 'w', append 'a' or exclusive creation 'x' mode.

We need to be careful with the 'w' mode as it will overwrite into the file if it already exists. All previous data are erased.

Writing a string or sequence of bytes (for binary files) is done using write() method. This method returns the number of characters written to the file.

with open("test.txt",'w',encoding = 'utf-8') as f:

f.write("my first file\n")

f.write("This file\n\n")

f.write("contains three lines\n")

This program will create a new file named 'test.txt' if it does not exist. If it does exist, it is overwritten.

We must include the newline characters ourselves to distinguish different lines.

How to read files in Python?

To read a file in Python, we must open the file in reading mode.

There are various methods available for this purpose. We can use the read(size) method to read in size number of data. If size parameter is not specified, it reads and returns up to the end of the file.

>>> f = open("test.txt",'r',encoding = 'utf-8')

>>> f.read(4) # read the first 4 data

'This'

>>> f.read(4) # read the next 4 data

' is '

>>> f.read() # read in the rest till end of file

'my first file\nThis file\ncontains three lines\n'

>>> f.read() # further reading returns empty sting

''

We can see that, the read() method returns newline as '\n'. Once the end of file is reached, we get empty string on further reading.

We can change our current file cursor (position) using the seek() method. Similarly, the tell() method returns our current position (in number of bytes).

>>> f.tell() # get the current file position

56

>>> f.seek(0) # bring file cursor to initial position

0

>>> print(f.read()) # read the entire file

This is my first file

This file

contains three lines

We can read a file line-by-line using a for loop. This is both efficient and fast.

>>> for line in f:

... print(line, end = '')

...

This is my first file

This file

contains three lines

The lines in file itself has a newline character '\n'.

Moreover, the print() end parameter to avoid two newlines when printing.

Alternately, we can use readline() method to read individual lines of a file. This method reads a file till the newline, including the newline character.

>>> f.readline()

'This is my first file\n'

>>> f.readline()

'This file\n'

>>> f.readline()

'contains three lines\n'

>>> f.readline()

''

Lastly, the readlines() method returns a list of remaining lines of the entire file. All these reading method return empty values when end of file (EOF) is reached.

>>> f.readlines()

['This is my first file\n', 'This file\n', 'contains three lines\n']

Python File Methods

There are various methods available with the file object. Some of them have been used in above examples.

Here is the complete list of methods in text mode with a brief description.

Python File Methods

Method Description

close() Close an open file. It has no effect if the file is already closed.

detach() Separate the underlying binary buffer from the TextIOBase and return it.

fileno() Return an integer number (file descriptor) of the file.

flush() Flush the write buffer of the file stream.

isatty() Return True if the file stream is interactive.

read(n) Read atmost n characters form the file. Reads till end of file if it is negative or None.

readable() Returns True if the file stream can be read from.

readline(n=-1) Read and return one line from the file. Reads in at most n bytes if specified.

readlines(n=-1) Read and return a list of lines from the file. Reads in at most n bytes/characters if specified.

seek(offset,from=SEEK\_SET) Change the file position to offset bytes, in reference to from (start, current, end).

seekable() Returns True if the file stream supports random access.

tell() Returns the current file location.

truncate(size=None) Resize the file stream to size bytes. If size is not specified, resize to current location.

writable() Returns True if the file stream can be written to.

write(s) Write string s to the file and return the number of characters written.

writelines(lines) Write a list of lines to the file.

**Os Module**:

Python Directory and Files Management

**In this article, you'll learn about file and directory management in Python, i.e. creating a directory, renaming it, listing all directories and working with them.**

If there are a large number of [files to handle](https://www.programiz.com/python-programming/file-operation) in your Python program, you can arrange your code within different directories to make things more manageable.

A directory or folder is a collection of files and sub directories. Python has the os [module](https://www.programiz.com/python-programming/modules), which provides us with many useful methods to work with directories (and files as well).

**Get Current Directory**

We can get the present working directory using the getcwd() method.

This method returns the current working directory in the form of a string. We can also use the getcwdb() method to get it as bytes object.

>>> import os

>>> os.getcwd()

'C:\\Program Files\\PyScripter'

>>> os.getcwdb()

b'C:\\Program Files\\PyScripter'

The extra backslash implies escape sequence. The print() function will render this properly.

>>> print(os.getcwd())

C:\Program Files\PyScripter

**Changing Directory**

We can change the current working directory using the chdir() method.

The new path that we want to change to must be supplied as a string to this method. We can use both forward slash (/) or the backward slash (\) to separate path elements.

It is safer to use escape sequence when using the backward slash.

>>> os.chdir('C:\\Python33')

>>> print(os.getcwd())

C:\Python33

**List Directories and Files**

All files and sub directories inside a directory can be known using the listdir() method.

This method takes in a path and returns a list of sub directories and files in that path. If no path is specified, it returns from the current working directory.

>>> print(os.getcwd())

C:\Python33

>>> os.listdir()

['DLLs',

'Doc',

'include',

'Lib',

'libs',

'LICENSE.txt',

'NEWS.txt',

'python.exe',

'pythonw.exe',

'README.txt',

'Scripts',

'tcl',

'Tools']

>>> os.listdir('G:\\')

['$RECYCLE.BIN',

'Movies',

'Music',

'Photos',

'Series',

'System Volume Information']

**Making a New Directory**

We can make a new directory using the mkdir() method.

This method takes in the path of the new directory. If the full path is not specified, the new directory is created in the current working directory.

>>> os.mkdir('test')

>>> os.listdir()

['test']

**Renaming a Directory or a File**

The rename() method can rename a directory or a file.

The first argument is the old name and the new name must be supplies as the second argument.

>>> os.listdir()

['test']

>>> os.rename('test','new\_one')

>>> os.listdir()

['new\_one']

**Removing Directory or File**

A file can be removed (deleted) using the remove() method.

Similarly, the rmdir() method removes an empty directory.

>>> os.listdir()

['new\_one', 'old.txt']

>>> os.remove('old.txt')

>>> os.listdir()

['new\_one']

>>> os.rmdir('new\_one')

>>> os.listdir()

[]

However, note that rmdir() method can only remove empty directories.

In order to remove a non-empty directory we can use the rmtree() method inside the shutil module.

>>> os.listdir()

['test']

>>> os.rmdir('test')

Traceback (most recent call last):

...

OSError: [WinError 145] The directory is not empty: 'test'

>>> import shutil

>>> shutil.rmtree('test')

>>> os.listdir()

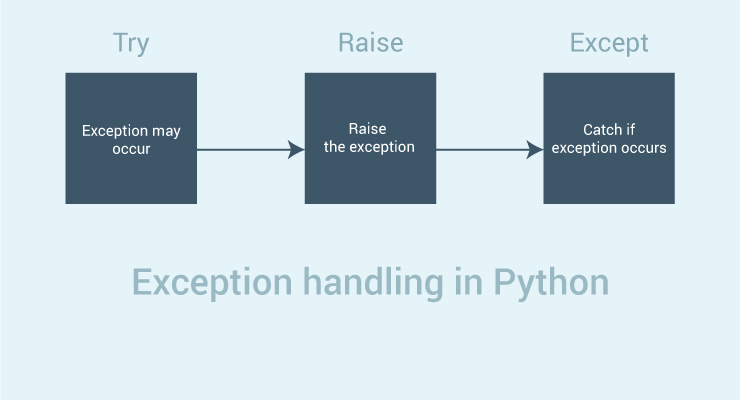
[]

**Exceptions**

|  |  |
| --- | --- |
| Python Built-in Exceptions | |
| Exception | Cause of Error |
| AssertionError | Raised when assert statement fails. |
| AttributeError | Raised when attribute assignment or reference fails. |
| EOFError | Raised when the input() functions hits end-of-file condition. |
| FloatingPointError | Raised when a floating point operation fails. |
| GeneratorExit | Raise when a generator's close() method is called. |
| ImportError | Raised when the imported module is not found. |
| IndexError | Raised when index of a sequence is out of range. |
| KeyError | Raised when a key is not found in a dictionary. |
| KeyboardInterrupt | Raised when the user hits interrupt key (Ctrl+c or delete). |
| MemoryError | Raised when an operation runs out of memory. |
| NameError | Raised when a variable is not found in local or global scope. |
| NotImplementedError | Raised by abstract methods. |
| OSError | Raised when system operation causes system related error. |
| OverflowError | Raised when result of an arithmetic operation is too large to be represented. |
| ReferenceError | Raised when a weak reference proxy is used to access a garbage collected referent. |
| RuntimeError | Raised when an error does not fall under any other category. |
|  |  |
| StopIteration | Raised by next() function to indicate that there is no further item to be returned by iterator. |
| SyntaxError | Raised by parser when syntax error is encountered. |
| IndentationError | Raised when there is incorrect indentation. |
| TabError | Raised when indentation consists of inconsistent tabs and spaces. |
| SystemError | Raised when interpreter detects internal error. |
| SystemExit | Raised by sys.exit() function. |
| TypeError | Raised when a function or operation is applied to an object of incorrect type. |
| UnboundLocalError | Raised when a reference is made to a local variable in a function or method, but no value has been bound to that variable. |
| UnicodeError | Raised when a Unicode-related encoding or decoding error occurs. |
| UnicodeEncodeError | Raised when a Unicode-related error occurs during encoding. |
| UnicodeDecodeError | Raised when a Unicode-related error occurs during decoding. |
| UnicodeTranslateError | Raised when a Unicode-related error occurs during translating. |
| ValueError | Raised when a function gets argument of correct type but improper value. |
| ZeroDivisionError | Raised when second operand of division or modulo operation is zero. |

# Python Exception Handling - Try, Except and Finally

**In this article, you'll learn how to handle exceptions in your Python program using try, except and finally statements. This will motivate you to write clean, readable and efficient code in Python.**



Python has many [built-in exceptions](https://www.programiz.com/python-programming/exceptions) which forces your program to output an error when something in it goes wrong.

When these exceptions occur, it causes the current process to stop and passes it to the calling process until it is handled. If not handled, our program will crash.

For example, if [function](https://www.programiz.com/python-programming/function) A calls function B which in turn calls function C and an exception occurs in function C. If it is not handled in C, the exception passes to B and then to A.

If never handled, an error message is spit out and our program come to a sudden, unexpected halt.

**Python Objects and Class**

In this article, you'll learn about the core functionality of Python, Python objects and classes. You'll learn what a class is, how to create it and use it in your program.

Python classes and objects

Python is an object oriented programming language. Unlike procedure oriented programming, where the main emphasis is on functions, object oriented programming stress on objects.

Object is simply a collection of data (variables) and methods (functions) that act on those data. And, class is a blueprint for the object.

We can think of class as a sketch (prototype) of a house. It contains all the details about the floors, doors, windows etc. Based on these descriptions we build the house. House is the object.

As, many houses can be made from a description, we can create many objects from a class. An object is also called an instance of a class and the process of creating this object is called instantiation.

**Defining a Class in Python**

Like function definitions begin with the keyword def, in Python, we define a class using the keyword class.

The first string is called docstring and has a brief description about the class. Although not mandatory, this is recommended.

Here is a simple class definition.

class MyNewClass:

'''This is a docstring. I have created a new class'''

pass

A class creates a new local namespace where all its attributes are defined. Attributes may be data or functions.

There are also special attributes in it that begins with double underscores (\_\_). For example, \_\_doc\_\_ gives us the docstring of that class.

As soon as we define a class, a new class object is created with the same name. This class object allows us to access the different attributes as well as to instantiate new objects of that class.

class MyClass:

"This is my second class"

a = 10

def func(self):

print('Hello')

# Output: 10

print(MyClass.a)

# Output: <function MyClass.func at 0x0000000003079BF8>

print(MyClass.func)

# Output: 'This is my second class'

print(MyClass.\_\_doc\_\_)

When you run the program, the output will be:

10

<function 0x7feaa932eae8="" at="" myclass.func="">

This is my second class

Creating an Object in Python

We saw that the class object could be used to access different attributes.

It can also be used to create new object instances (instantiation) of that class. The procedure to create an object is similar to a function call.

>>> ob = MyClass()

This will create a new instance object named ob. We can access attributes of objects using the object name prefix.

Attributes may be data or method. Method of an object are corresponding functions of that class. Any function object that is a class attribute defines a method for objects of that class.

This means to say, since MyClass.func is a function object (attribute of class), ob.func will be a method object.

class MyClass:

"This is my second class"

a = 10

def func(self):

print('Hello')

# create a new MyClass

ob = MyClass()

# Output: <function MyClass.func at 0x000000000335B0D0>

print(MyClass.func)

# Output: <bound method MyClass.func of <\_\_main\_\_.MyClass object at 0x000000000332DEF0>>

print(ob.func)

# Calling function func()

# Output: Hello

ob.func()

You may have noticed the self parameter in function definition inside the class but, we called the method simply as ob.func() without any arguments. It still worked.

This is because, whenever an object calls its method, the object itself is passed as the first argument. So, ob.func() translates into MyClass.func(ob).

In general, calling a method with a list of n arguments is equivalent to calling the corresponding function with an argument list that is created by inserting the method's object before the first argument.

For these reasons, the first argument of the function in class must be the object itself. This is conventionally called self. It can be named otherwise but we highly recommend to follow the convention.

Now you must be familiar with class object, instance object, function object, method object and their differences.

**Constructors in Python**

Class functions that begins with double underscore (\_\_) are called special functions as they have special meaning.

Of one particular interest is the \_\_init\_\_() function. This special function gets called whenever a new object of that class is instantiated.

This type of function is also called constructors in Object Oriented Programming (OOP). We normally use it to initialize all the variables.

class ComplexNumber:

def \_\_init\_\_(self,r = 0,i = 0):

self.real = r

self.imag = i

def getData(self):

print("{0}+{1}j".format(self.real,self.imag))

# Create a new ComplexNumber object

c1 = ComplexNumber(2,3)

# Call getData() function

# Output: 2+3j

c1.getData()

# Create another ComplexNumber object

# and create a new attribute 'attr'

c2 = ComplexNumber(5)

c2.attr = 10

# Output: (5, 0, 10)

print((c2.real, c2.imag, c2.attr))

# but c1 object doesn't have attribute 'attr'

# AttributeError: 'ComplexNumber' object has no attribute 'attr'

c1.attr

In the above example, we define a new class to represent complex numbers. It has two functions, \_\_init\_\_() to initialize the variables (defaults to zero) and getData() to display the number properly.

An interesting thing to note in the above step is that attributes of an object can be created on the fly. We created a new attribute attr for object c2 and we read it as well. But this did not create that attribute for object c1.

Deleting Attributes and Objects

Any attribute of an object can be deleted anytime, using the del statement. Try the following on the Python shell to see the output.

>>> c1 = ComplexNumber(2,3)

>>> del c1.imag

>>> c1.getData()

Traceback (most recent call last):

...

AttributeError: 'ComplexNumber' object has no attribute 'imag'

>>> del ComplexNumber.getData

>>> c1.getData()

Traceback (most recent call last):

...

AttributeError: 'ComplexNumber' object has no attribute 'getData'

We can even delete the object itself, using the del statement.

>>> c1 = ComplexNumber(1,3)

>>> del c1

>>> c1

Traceback (most recent call last):

...

NameError: name 'c1' is not defined

Actually, it is more complicated than that. When we do c1 = ComplexNumber(1,3), a new instance object is created in memory and the name c1 binds with it.

On the command del c1, this binding is removed and the name c1 is deleted from the corresponding namespace. The object however continues to exist in memory and if no other name is bound to it, it is later automatically destroyed.

This automatic destruction of unreferenced objects in Python is also called garbage collection.

**Python Inheritance**

Inheritance enable us to define a class that takes all the functionality from parent class and allows us to add more. In this article, you will learn to use inheritance in Python.

Creating derived class from a base class using inheritance

Inheritance is a powerful feature in object oriented programming.

It refers to defining a new class with little or no modification to an existing class. The new class is called derived (or child) class and the one from which it inherits is called the base (or parent) class.

Python Inheritance Syntax:

class BaseClass:

Body of base class

class DerivedClass(BaseClass):

Body of derived class

Derived class inherits features from the base class, adding new features to it. This results into re-usability of code.

Example of Inheritance in Python

To demonstrate the use of inheritance, let us take an example.

A polygon is a closed figure with 3 or more sides. Say, we have a class called Polygon defined as follows.

class Polygon:

def \_\_init\_\_(self, no\_of\_sides):

self.n = no\_of\_sides

self.sides = [0 for i in range(no\_of\_sides)]

def inputSides(self):

self.sides = [float(input("Enter side "+str(i+1)+" : ")) for i in range(self.n)]

def dispSides(self):

for i in range(self.n):

print("Side",i+1,"is",self.sides[i])

This class has data attributes to store the number of sides, n and magnitude of each side as a list, sides.

Method inputSides() takes in magnitude of each side and similarly, dispSides() will display these properly.

A triangle is a polygon with 3 sides. So, we can created a class called Triangle which inherits from Polygon. This makes all the attributes available in class Polygon readily available in Triangle. We don't need to define them again (code re-usability). Triangle is defined as follows.

class Triangle(Polygon):

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

Polygon.\_\_init\_\_(self,3)

def findArea(self):

a, b, c = self.sides

# calculate the semi-perimeter

s = (a + b + c) / 2

area = (s\*(s-a)\*(s-b)\*(s-c)) \*\* 0.5

print('The area of the triangle is %0.2f' %area)

However, class Triangle has a new method findArea() to find and print the area of the triangle. Here is a sample run.

>>> t = Triangle()

>>> t.inputSides()

Enter side 1 : 3

Enter side 2 : 5

Enter side 3 : 4

>>> t.dispSides()

Side 1 is 3.0

Side 2 is 5.0

Side 3 is 4.0

>>> t.findArea()

The area of the triangle is 6.00

We can see that, even though we did not define methods like inputSides() or dispSides() for class Triangle, we were able to use them.

If an attribute is not found in the class, search continues to the base class. This repeats recursively, if the base class is itself derived from other classes.

**Method Overriding in Python**

In the above example, notice that \_\_init\_\_() method was defined in both classes, Triangle as well Polygon. When this happens, the method in the derived class overrides that in the base class. This is to say, \_\_init\_\_() in Triangle gets preference over the same in Polygon.

Generally when overriding a base method, we tend to extend the definition rather than simply replace it. The same is being done by calling the method in base class from the one in derived class (calling Polygon.\_\_init\_\_() from \_\_init\_\_() in Triangle).

A better option would be to use the built-in function super(). So, super().\_\_init\_\_(3) is equivalent to Polygon.\_\_init\_\_(self,3) and is preferred. You can learn more about the super() function in Python.

Two built-in functions isinstance() and issubclass() are used to check inheritances. Function isinstance() returns True if the object is an instance of the class or other classes derived from it. Each and every class in Python inherits from the base class object.

>>> isinstance(t,Triangle)

True

>>> isinstance(t,Polygon)

True

>>> isinstance(t,int)

False

>>> isinstance(t,object)

True

Similarly, issubclass() is used to check for class inheritance.

>>> issubclass(Polygon,Triangle)

False

>>> issubclass(Triangle,Polygon)

True

>>> issubclass(bool,int)

True

Python Multiple Inheritance

In this article, you'll learn what is multiple inheritance in Python and how to use it in your program. You'll also learn about multilevel inheritance and the method resolution order.

Python Multiple Inheritance

Like C++, a class can be derived from more than one base classes in Python. This is called multiple inheritance.

In multiple inheritance, the features of all the base classes are inherited into the derived class. The syntax for multiple inheritance is similar to single inheritance.

Python Multiple Inheritance Example

class Base1:

pass

class Base2:

pass

class MultiDerived(Base1, Base2):

pass

Here, MultiDerived is derived from classes Base1 and Base2.

Multiple Inheritance in Python

The class MultiDerived inherits from both Base1 and Base2.

Multilevel Inheritance in Python

On the other hand, we can also inherit form a derived class. This is called multilevel inheritance. It can be of any depth in Python.

In multilevel inheritance, features of the base class and the derived class is inherited into the new derived class.

An example with corresponding visualization is given below.

class Base:

pass

class Derived1(Base):

pass

class Derived2(Derived1):

pass

Here, Derived1 is derived from Base, and Derived2 is derived from Derived1.

**Python Iterators**

Iterators are objects that can be iterated upon. In this tutorial, you will learn how iterator works and how you can build your own iterator using \_\_iter\_\_ and \_\_next\_\_ methods.

Python Iterators

Iterators are everywhere in Python. They are elegantly implemented within for loops, comprehensions, generators etc. but hidden in plain sight.

Iterator in Python is simply an object that can be iterated upon. An object which will return data, one element at a time.

Technically speaking, Python iterator object must implement two special methods, \_\_iter\_\_() and \_\_next\_\_(), collectively called the iterator protocol.

An object is called iterable if we can get an iterator from it. Most of built-in containers in Python like: list, tuple, string etc. are iterables.

The iter() function (which in turn calls the \_\_iter\_\_() method) returns an iterator from them.

Iterating Through an Iterator in Python

We use the next() function to manually iterate through all the items of an iterator. When we reach the end and there is no more data to be returned, it will raise StopIteration. Following is an example.

# define a list

my\_list = [4, 7, 0, 3]

# get an iterator using iter()

my\_iter = iter(my\_list)

## iterate through it using next()

#prints 4

print(next(my\_iter))

#prints 7

print(next(my\_iter))

## next(obj) is same as obj.\_\_next\_\_()

#prints 0

print(my\_iter.\_\_next\_\_())

#prints 3

print(my\_iter.\_\_next\_\_())

## This will raise error, no items left

next(my\_iter)

A more elegant way of automatically iterating is by using the for loop. Using this, we can iterate over any object that can return an iterator, for example list, string, file etc.

>>> for element in my\_list:

... print(element)

...

4

7

0

3

How for loop actually works?

As we see in the above example, the for loop was able to iterate automatically through the list.

In fact the for loop can iterate over any iterable. Let's take a closer look at how the for loop is actually implemented in Python.

for element in iterable:

# do something with element

Is actually implemented as.

# create an iterator object from that iterable

iter\_obj = iter(iterable)

# infinite loop

while True:

try:

# get the next item

element = next(iter\_obj)

# do something with element

except StopIteration:

# if StopIteration is raised, break from loop

break

So internally, the for loop creates an iterator object, iter\_obj by calling iter() on the iterable.

Ironically, this for loop is actually an infinite while loop.

Inside the loop, it calls next() to get the next element and executes the body of the for loop with this value. After all the items exhaust, StopIteration is raised which is internally caught and the loop ends. Note that any other kind of exception will pass through.

Building Your Own Iterator in Python

Building an iterator from scratch is easy in Python. We just have to implement the methods \_\_iter\_\_() and \_\_next\_\_().

The \_\_iter\_\_() method returns the iterator object itself. If required, some initialization can be performed.

The \_\_next\_\_() method must return the next item in the sequence. On reaching the end, and in subsequent calls, it must raise StopIteration.

Here, we show an example that will give us next power of 2 in each iteration. Power exponent starts from zero up to a user set number.

class PowTwo:

"""Class to implement an iterator

of powers of two"""

def \_\_init\_\_(self, max = 0):

self.max = max

def \_\_iter\_\_(self):

self.n = 0

return self

def \_\_next\_\_(self):

if self.n <= self.max:

result = 2 \*\* self.n

self.n += 1

return result

else:

raise StopIteration

Now we can create an iterator and iterate through it as follows.

>>> a = PowTwo(4)

>>> i = iter(a)

>>> next(i)

1

>>> next(i)

2

>>> next(i)

4

>>> next(i)

8

>>> next(i)

16

>>> next(i)

Traceback (most recent call last):

...

StopIteration

We can also use a for loop to iterate over our iterator class.

>>> for i in PowTwo(5):

... print(i)

...

32

Python Infinite Iterators

It is not necessary that the item in an iterator object has to exhaust. There can be infinite iterators (which never ends). We must be careful when handling such iterator.

Here is a simple example to demonstrate infinite iterators.

The built-in function iter() can be called with two arguments where the first argument must be a callable object (function) and second is the sentinel. The iterator calls this function until the returned value is equal to the sentinel.

>>> int()

0

>>> inf = iter(int,1)

>>> next(inf)

0

>>> next(inf)

0

We can see that the int() function always returns 0. So passing it as iter(int,1) will return an iterator that calls int() until the returned value equals 1. This never happens and we get an infinite iterator.

We can also built our own infinite iterators. The following iterator will, theoretically, return all the odd numbers.

class InfIter:

"""Infinite iterator to return all

odd numbers"""

def \_\_iter\_\_(self):

self.num = 1

return self

def \_\_next\_\_(self):

num = self.num

self.num += 2

return num

A sample run would be as follows.

>>> a = iter(InfIter())

>>> next(a)

1

>>> next(a)

3

>>> next(a)

5

>>> next(a)

7

And so on...

Be careful to include a terminating condition, when iterating over these type of infinite iterators.

The advantage of using iterators is that they save resources. Like shown above, we could get all the odd numbers without storing the entire number system in memory. We can have infinite items (theoretically) in finite memory.

Iterator also makes our code look cool.

There's an easier way to create iterators in Python. To learn more visit: Python generators using yield.

**Python Generators**

In this article, you'll learn how to create iterations easily using Python generators, how is it different from iterators and normal functions, and why you should use it.

Python Generators using Yield

There is a lot of overhead in building an iterator in Python; we have to implement a class with \_\_iter\_\_() and \_\_next\_\_() method, keep track of internal states, raise StopIteration when there was no values to be returned etc.

This is both lengthy and counter intuitive. Generator comes into rescue in such situations.

Python generators are a simple way of creating iterators. All the overhead we mentioned above are automatically handled by generators in Python.

Simply speaking, a generator is a function that returns an object (iterator) which we can iterate over (one value at a time).

How to create a generator in Python?

It is fairly simple to create a generator in Python. It is as easy as defining a normal function with yield statement instead of a return statement.

If a function contains at least one yield statement (it may contain other yield or return statements), it becomes a generator function. Both yield and return will return some value from a function.

The difference is that, while a return statement terminates a function entirely, yield statement pauses the function saving all its states and later continues from there on successive calls.

Differences between Generator function and a Normal function

Here is how a generator function differs from a normal function.

Generator function contains one or more yield statement.

When called, it returns an object (iterator) but does not start execution immediately.

Methods like \_\_iter\_\_() and \_\_next\_\_() are implemented automatically. So we can iterate through the items using next().

Once the function yields, the function is paused and the control is transferred to the caller.

Local variables and their states are remembered between successive calls.

Finally, when the function terminates, StopIteration is raised automatically on further calls

# A simple generator function

def my\_gen():

n = 1

print('This is printed first')

# Generator function contains yield statements

yield n

n += 1

print('This is printed second')

yield n

n += 1

print('This is printed at last')

yield n

An interactive run in the interpreter is given below. Run these in the Python shell to see the output.

>>> # It returns an object but does not start execution immediately.

>>> a = my\_gen()

>>> # We can iterate through the items using next().

>>> next(a)

This is printed first

1

>>> # Once the function yields, the function is paused and the control is transferred to the caller.

>>> # Local variables and theirs states are remembered between successive calls.

>>> next(a)

This is printed second

2

>>> next(a)

This is printed at last

3

>>> # Finally, when the function terminates, StopIteration is raised automatically on further calls.

>>> next(a)

Traceback (most recent call last):

...

StopIteration

>>> next(a)

Traceback (most recent call last):

...

StopIteration

One interesting thing to note in the above example is that, the value of variable n is remembered between each call.

Unlike normal functions, the local variables are not destroyed when the function yields. Furthermore, the generator object can be iterated only once.

To restart the process we need to create another generator object using something like a = my\_gen().

Note: One final thing to note is that we can use generators with for loops directly.

This is because, a for loop takes an iterator and iterates over it using next() function. It automatically ends when StopIteration is raised. Check here to know how a for loop is actually implemented in Python.

# A simple generator function

def my\_gen():

n = 1

print('This is printed first')

# Generator function contains yield statements

yield n

n += 1

print('This is printed second')

yield n

n += 1

print('This is printed at last')

yield n

# Using for loop

for item in my\_gen():

print(item)

When you run the program, the output will be:

This is printed first

1

This is printed second

2

This is printed at last

3

Python Generators with a Loop

The above example is of less use and we studied it just to get an idea of what was happening in the background.

Normally, generator functions are implemented with a loop having a suitable terminating condition.

Let's take an example of a generator that reverses a string.

def rev\_str(my\_str):

length = len(my\_str)

for i in range(length - 1,-1,-1):

yield my\_str[i]

# For loop to reverse the string

# Output:

# o

# l

# l

# e

# h

for char in rev\_str("hello"):

print(char)

In this example, we use range() function to get the index in reverse order using the for loop.

It turns out that this generator function not only works with string, but also with other kind of iterables like list, tuple etc.

**Python Generator Expression**

Simple generators can be easily created on the fly using generator expressions. It makes building generators easy.

Same as lambda function creates an anonymous function, generator expression creates an anonymous generator function.

The syntax for generator expression is similar to that of a list comprehension in Python. But the square brackets are replaced with round parentheses.

The major difference between a list comprehension and a generator expression is that while list comprehension produces the entire list, generator expression produces one item at a time.

They are kind of lazy, producing items only when asked for. For this reason, a generator expression is much more memory efficient than an equivalent list comprehension.

# Initialize the list

my\_list = [1, 3, 6, 10]

# square each term using list comprehension

# Output: [1, 9, 36, 100]

[x\*\*2 for x in my\_list]

# same thing can be done using generator expression

# Output: <generator object <genexpr> at 0x0000000002EBDAF8>

(x\*\*2 for x in my\_list)

We can see above that the generator expression did not produce the required result immediately. Instead, it returned a generator object with produces items on demand.

# Intialize the list

my\_list = [1, 3, 6, 10]

a = (x\*\*2 for x in my\_list)

# Output: 1

print(next(a))

# Output: 9

print(next(a))

# Output: 36

print(next(a))

# Output: 100

print(next(a))

# Output: StopIteration

next(a)

Generator expression can be used inside functions. When used in such a way, the round parentheses can be dropped.

>>> sum(x\*\*2 for x in my\_list)

146

>>> max(x\*\*2 for x in my\_list)

100

Why generators are used in Python?

There are several reasons which make generators an attractive implementation to go for.

1. Easy to Implement

Generators can be implemented in a clear and concise way as compared to their iterator class counterpart. Following is an example to implement a sequence of power of 2's using iterator class.

class PowTwo:

def \_\_init\_\_(self, max = 0):

self.max = max

def \_\_iter\_\_(self):

self.n = 0

return self

def \_\_next\_\_(self):

if self.n > self.max:

raise StopIteration

result = 2 \*\* self.n

self.n += 1

return result

This was lengthy. Now lets do the same using a generator function.

def PowTwoGen(max = 0):

n = 0

while n < max:

yield 2 \*\* n

n += 1

Since, generators keep track of details automatically, it was concise and much cleaner in implementation.

2. Memory Efficient

A normal function to return a sequence will create the entire sequence in memory before returning the result. This is an overkill if the number of items in the sequence is very large.

Generator implementation of such sequence is memory friendly and is preferred since it only produces one item at a time.

3. Represent Infinite Stream

Generators are excellent medium to represent an infinite stream of data. Infinite streams cannot be stored in memory and since generators produce only one item at a time, it can represent infinite stream of data.

The following example can generate all the even numbers (at least in theory).

def all\_even():

n = 0

while True:

yield n

n += 2

4. Pipelining Generators

Generators can be used to pipeline a series of operations. This is best illustrated using an example.

Suppose we have a log file from a famous fast food chain. The log file has a column (4th column) that keeps track of the number of pizza sold every hour and we want to sum it to find the total pizzas sold in 5 years.

Assume everything is in string and numbers that are not available are marked as 'N/A'. A generator implementation of this could be as follows.

with open('sells.log') as file:

pizza\_col = (line[3] for line in file)

per\_hour = (int(x) for x in pizza\_col if x != 'N/A')

print("Total pizzas sold = ",sum(per\_hour))

This pipelining is efficient and easy to read (and yes, a lot cooler!)

**Python Decorators**

A decorator takes in a function, adds some functionality and returns it. In this article, you will learn how you can create a decorator and why you should use it.

Python Decorator

Python has an interesting feature called decorators to add functionality to an existing code.

This is also called metaprogramming as a part of the program tries to modify another part of the program at compile time.

What you need to know before learning decorators?

In order to understand about decorators, we must first know a few basic things in Python.

We must be comfortable with the fact that, everything in Python (Yes! Even classes), are objects. Names that we define are simply identifiers bound to these objects. Functions are no exceptions, they are objects too (with attributes). Various different names can be bound to the same function object.

Here is an example.

def first(msg):

print(msg)

first("Hello")

second = first

second("Hello")

When you run the code, both functions first and second gives same output. Here, the names first and second refer to the same function object.

Now things start getting weirder.

Functions can be passed as arguments to another function.

If you have used functions like map, filter and reduce in Python, then you already know about this.

Such function that take other functions as arguments are also called higher order functions. Here is an example of such a function.

def inc(x):

return x + 1

def dec(x):

return x - 1

def operate(func, x):

result = func(x)

return result

We invoke the function as follows.

>>> operate(inc,3)

4

>>> operate(dec,3)

2

Furthermore, a function can return another function.

def is\_called():

def is\_returned():

print("Hello")

return is\_returned

new = is\_called()

#Outputs "Hello"

new()

Here, is\_returned() is a nested function which is defined and returned, each time we call is\_called().

Finally, we must know about closures in Python.

Getting back To decorators

Functions and methods are called callable as they can be called.

In fact, any object which implements the special method \_\_call\_\_() is termed callable. So, in the most basic sense, a decorator is a callable that returns a callable.

Basically, a decorator takes in a function, adds some functionality and returns it.

def make\_pretty(func):

def inner():

print("I got decorated")

func()

return inner

def ordinary():

print("I am ordinary")

When you run the following codes in shell,

>>> ordinary()

I am ordinary

>>> # let's decorate this ordinary function

>>> pretty = make\_pretty(ordinary)

>>> pretty()

I got decorated

I am ordinary

In the example shown above, make\_pretty() is a decorator. In the assignment step.

pretty = make\_pretty(ordinary)

The function ordinary() got decorated and the returned function was given the name pretty.

We can see that the decorator function added some new functionality to the original function. This is similar to packing a gift. The decorator acts as a wrapper. The nature of the object that got decorated (actual gift inside) does not alter. But now, it looks pretty (since it got decorated).

Generally, we decorate a function and reassign it as,

ordinary = make\_pretty(ordinary).

This is a common construct and for this reason, Python has a syntax to simplify this.

We can use the @ symbol along with the name of the decorator function and place it above the definition of the function to be decorated. For example,

@make\_pretty

def ordinary():

print("I am ordinary")

is equivalent to

def ordinary():

print("I am ordinary")

ordinary = make\_pretty(ordinary)

This is just a syntactic sugar to implement decorators.

Decorating Functions with Parameters

The above decorator was simple and it only worked with functions that did not have any parameters. What if we had functions that took in parameters like below?

def divide(a, b):

return a/b

This function has two parameters, a and b. We know, it will give error if we pass in b as 0.

>>> divide(2,5)

0.4

>>> divide(2,0)

Traceback (most recent call last):

...

ZeroDivisionError: division by zero

Now let's make a decorator to check for this case that will cause the error.

def smart\_divide(func):

def inner(a,b):

print("I am going to divide",a,"and",b)

if b == 0:

print("Whoops! cannot divide")

return

return func(a,b)

return inner

@smart\_divide

def divide(a,b):

return a/b

This new implementation will return None if the error condition arises.

>>> divide(2,5)

I am going to divide 2 and 5

0.4

>>> divide(2,0)

I am going to divide 2 and 0

Whoops! cannot divide

In this manner we can decorate functions that take parameters.

A keen observer will notice that parameters of the nested inner() function inside the decorator is same as the parameters of functions it decorates. Taking this into account, now we can make general decorators that work with any number of parameter.

In Python, this magic is done as function(\*args, \*\*kwargs). In this way, args will be the tuple of positional arguments and kwargs will be the dictionary of keyword arguments. An example of such decorator will be.

def works\_for\_all(func):

def inner(\*args, \*\*kwargs):

print("I can decorate any function")

return func(\*args, \*\*kwargs)

return inner

Chaining Decorators in Python

Multiple decorators can be chained in Python.

This is to say, a function can be decorated multiple times with different (or same)

def star(func):

def inner(\*args, \*\*kwargs):

print("\*" \* 30)

func(\*args, \*\*kwargs)

print("\*" \* 30)

return inner

def percent(func):

def inner(\*args, \*\*kwargs):

print("%" \* 30)

func(\*args, \*\*kwargs)

print("%" \* 30)

return inner

@star

@percent

def printer(msg):

print(msg)

printer("Hello")

This will give the output.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Hello

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

The above syntax of,

@star

@percent

def printer(msg):

print(msg)

is equivalent to

def printer(msg):

print(msg)

printer = star(percent(printer))

The order in which we chain decorators matter. If we had reversed the order as,

@percent

@star

def printer(msg):

print(msg)

The execution would take place as,

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Hello

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

**Practice Questions and Answers:**

Python practise questions:

Question 1

Level 1

Question:

Write a program which will find all such numbers which are divisible by 7 but are not a multiple of 5,

between 2000 and 3200 (both included).

The numbers obtained should be printed in a comma-separated sequence on a single line.

Hints:

Consider use range(#begin, #end) method

Solution:

l=[]

for i in range(2000, 3201):

if (i%7==0) and (i%5!=0):

l.append(str(i))

print ','.join(l)

#----------------------------------------#

#----------------------------------------#

Question 2

Level 1

Question:

Write a program which can compute the factorial of a given numbers.

The results should be printed in a comma-separated sequence on a single line.

Suppose the following input is supplied to the program:

8

Then, the output should be:

40320

Hints:

In case of input data being supplied to the question, it should be assumed to be a console input.

Solution:

def fact(x):

if x == 0:

return 1

return x \* fact(x - 1)

x=int(raw\_input())

print fact(x)

#----------------------------------------#

#----------------------------------------#

Question 3

Level 1

Question:

With a given integral number n, write a program to generate a dictionary that contains (i, i\*i) such that is an integral number between 1 and n (both included). and then the program should print the dictionary.

Suppose the following input is supplied to the program:

8

Then, the output should be:

{1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64}

Hints:

In case of input data being supplied to the question, it should be assumed to be a console input.

Consider use dict()

Solution:

n=int(raw\_input())

d=dict()

for i in range(1,n+1):

d[i]=i\*i

print d

#----------------------------------------#

#----------------------------------------#

Question 4

Level 1

Question:

Write a program which accepts a sequence of comma-separated numbers from console and generate a list and a tuple which contains every number.

Suppose the following input is supplied to the program:

34,67,55,33,12,98

Then, the output should be:

['34', '67', '55', '33', '12', '98']

('34', '67', '55', '33', '12', '98')

Hints:

In case of input data being supplied to the question, it should be assumed to be a console input.

tuple() method can convert list to tuple

Solution:

values=raw\_input()

l=values.split(",")

t=tuple(l)

print l

print t

#----------------------------------------#

#----------------------------------------#

Question 5

Level 1

Question:

Define a class which has at least two methods:

getString: to get a string from console input

printString: to print the string in upper case.

Also please include simple test function to test the class methods.

Hints:

Use \_\_init\_\_ method to construct some parameters

Solution:

class InputOutString(object):

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

self.s = ""

def getString(self):

self.s = raw\_input()

def printString(self):

print self.s.upper()

strObj = InputOutString()

strObj.getString()

strObj.printString()

#----------------------------------------#

#----------------------------------------#

Question 6

Level 2

Question:

Write a program that calculates and prints the value according to the given formula:

Q = Square root of [(2 \* C \* D)/H]

Following are the fixed values of C and H:

C is 50. H is 30.

D is the variable whose values should be input to your program in a comma-separated sequence.

Example

Let us assume the following comma separated input sequence is given to the program:

100,150,180

The output of the program should be:

18,22,24

Hints:

If the output received is in decimal form, it should be rounded off to its nearest value (for example, if the output received is 26.0, it should be printed as 26)

In case of input data being supplied to the question, it should be assumed to be a console input.

Solution:

#!/usr/bin/env python

import math

c=50

h=30

value = []

items=[x for x in raw\_input().split(',')]

for d in items:

value.append(str(int(round(math.sqrt(2\*c\*float(d)/h)))))

print ','.join(value)

#----------------------------------------#

#----------------------------------------#

Question 7

Level 2

Question:

Write a program which takes 2 digits, X,Y as input and generates a 2-dimensional array. The element value in the i-th row and j-th column of the array should be i\*j.

Note: i=0,1.., X-1; j=0,1,¡­Y-1.

Example

Suppose the following inputs are given to the program:

3,5

Then, the output of the program should be:

[[0, 0, 0, 0, 0], [0, 1, 2, 3, 4], [0, 2, 4, 6, 8]]

Hints:

Note: In case of input data being supplied to the question, it should be assumed to be a console input in a comma-separated form.

Solution:

input\_str = raw\_input()

dimensions=[int(x) for x in input\_str.split(',')]

rowNum=dimensions[0]

colNum=dimensions[1]

multilist = [[0 for col in range(colNum)] for row in range(rowNum)]

for row in range(rowNum):

for col in range(colNum):

multilist[row][col]= row\*col

print multilist

#----------------------------------------#

#----------------------------------------#

Question 8

Level 2

Question:

Write a program that accepts a comma separated sequence of words as input and prints the words in a comma-separated sequence after sorting them alphabetically.

Suppose the following input is supplied to the program:

without,hello,bag,world

Then, the output should be:

bag,hello,without,world

Hints:

In case of input data being supplied to the question, it should be assumed to be a console input.

Solution:

items=[x for x in raw\_input().split(',')]

items.sort()

print ','.join(items)

#----------------------------------------#

#----------------------------------------#

Question 9

Level 2

Question£º

Write a program that accepts sequence of lines as input and prints the lines after making all characters in the sentence capitalized.

Suppose the following input is supplied to the program:

Hello world

Practice makes perfect

Then, the output should be:

HELLO WORLD

PRACTICE MAKES PERFECT

Hints:

In case of input data being supplied to the question, it should be assumed to be a console input.

Solution:

lines = []

while True:

s = raw\_input()

if s:

lines.append(s.upper())

else:

break;

for sentence in lines:

print sentence

#----------------------------------------#

#----------------------------------------#

Question 10

Level 2

Question:

Write a program that accepts a sequence of whitespace separated words as input and prints the words after removing all duplicate words and sorting them alphanumerically.

Suppose the following input is supplied to the program:

hello world and practice makes perfect and hello world again

Then, the output should be:

again and hello makes perfect practice world

Hints:

In case of input data being supplied to the question, it should be assumed to be a console input.

We use set container to remove duplicated data automatically and then use sorted() to sort the data.

Solution:

s = raw\_input()

words = [word for word in s.split(" ")]

print " ".join(sorted(list(set(words))))

#----------------------------------------#

#----------------------------------------#

Question 11

Level 2

Question:

Write a program which accepts a sequence of comma separated 4 digit binary numbers as its input and then check whether they are divisible by 5 or not. The numbers that are divisible by 5 are to be printed in a comma separated sequence.

Example:

0100,0011,1010,1001

Then the output should be:

1010

Notes: Assume the data is input by console.

Hints:

In case of input data being supplied to the question, it should be assumed to be a console input.

Solution:

value = []

items=[x for x in raw\_input().split(',')]

for p in items:

intp = int(p, 2)

if not intp%5:

value.append(p)

print ','.join(value)

#----------------------------------------#

#----------------------------------------#

Question 12

Level 2

Question:

Write a program, which will find all such numbers between 1000 and 3000 (both included) such that each digit of the number is an even number.

The numbers obtained should be printed in a comma-separated sequence on a single line.

Hints:

In case of input data being supplied to the question, it should be assumed to be a console input.

Solution:

values = []

for i in range(1000, 3001):

s = str(i)

if (int(s[0])%2==0) and (int(s[1])%2==0) and (int(s[2])%2==0) and (int(s[3])%2==0):

values.append(s)

print ",".join(values)

#----------------------------------------#

#----------------------------------------#

Question 13

Level 2

Question:

Write a program that accepts a sentence and calculate the number of letters and digits.

Suppose the following input is supplied to the program:

hello world! 123

Then, the output should be:

LETTERS 10

DIGITS 3

Hints:

In case of input data being supplied to the question, it should be assumed to be a console input.

Solution:

s = raw\_input()

d={"DIGITS":0, "LETTERS":0}

for c in s:

if c.isdigit():

d["DIGITS"]+=1

elif c.isalpha():

d["LETTERS"]+=1

else:

pass

print "LETTERS", d["LETTERS"]

print "DIGITS", d["DIGITS"]

#----------------------------------------#

#----------------------------------------#

Question 14

Level 2

Question:

Write a program that accepts a sentence and calculate the number of upper case letters and lower case letters.

Suppose the following input is supplied to the program:

Hello world!

Then, the output should be:

UPPER CASE 1

LOWER CASE 9

Hints:

In case of input data being supplied to the question, it should be assumed to be a console input.

Solution:

s = raw\_input()

d={"UPPER CASE":0, "LOWER CASE":0}

for c in s:

if c.isupper():

d["UPPER CASE"]+=1

elif c.islower():

d["LOWER CASE"]+=1

else:

pass

print "UPPER CASE", d["UPPER CASE"]

print "LOWER CASE", d["LOWER CASE"]

#----------------------------------------#

#----------------------------------------#

Question 15

Level 2

Question:

Write a program that computes the value of a+aa+aaa+aaaa with a given digit as the value of a.

Suppose the following input is supplied to the program:

9

Then, the output should be:

11106

Hints:

In case of input data being supplied to the question, it should be assumed to be a console input.

Solution:

a = raw\_input()

n1 = int( "%s" % a )

n2 = int( "%s%s" % (a,a) )

n3 = int( "%s%s%s" % (a,a,a) )

n4 = int( "%s%s%s%s" % (a,a,a,a) )

print n1+n2+n3+n4

#----------------------------------------#

#----------------------------------------#

Question 16

Level 2

Question:

Use a list comprehension to square each odd number in a list. The list is input by a sequence of comma-separated numbers.

Suppose the following input is supplied to the program:

1,2,3,4,5,6,7,8,9

Then, the output should be:

1,3,5,7,9

Hints:

In case of input data being supplied to the question, it should be assumed to be a console input.

Solution:

values = raw\_input()

numbers = [x for x in values.split(",") if int(x)%2!=0]

print ",".join(numbers)

#----------------------------------------#

Question 17

Level 2

Question:

Write a program that computes the net amount of a bank account based a transaction log from console input. The transaction log format is shown as following:

D 100

W 200

¡­

D means deposit while W means withdrawal.

Suppose the following input is supplied to the program:

D 300

D 300

W 200

D 100

Then, the output should be:

500

Hints:

In case of input data being supplied to the question, it should be assumed to be a console input.

Solution:

import sys

netAmount = 0

while True:

s = raw\_input()

if not s:

break

values = s.split(" ")

operation = values[0]

amount = int(values[1])

if operation=="D":

netAmount+=amount

elif operation=="W":

netAmount-=amount

else:

pass

print netAmount

#----------------------------------------#

#----------------------------------------#

Question 18

Level 3

Question:

A website requires the users to input username and password to register. Write a program to check the validity of password input by users.

Following are the criteria for checking the password:

1. At least 1 letter between [a-z]

2. At least 1 number between [0-9]

1. At least 1 letter between [A-Z]

3. At least 1 character from [$#@]

4. Minimum length of transaction password: 6

5. Maximum length of transaction password: 12

Your program should accept a sequence of comma separated passwords and will check them according to the above criteria. Passwords that match the criteria are to be printed, each separated by a comma.

Example

If the following passwords are given as input to the program:

ABd1234@1,a F1#,2w3E\*,2We3345

Then, the output of the program should be:

ABd1234@1

Hints:

In case of input data being supplied to the question, it should be assumed to be a console input.

Solutions:

import re

value = []

items=[x for x in raw\_input().split(',')]

for p in items:

if len(p)<6 or len(p)>12:

continue

else:

pass

if not re.search("[a-z]",p):

continue

elif not re.search("[0-9]",p):

continue

elif not re.search("[A-Z]",p):

continue

elif not re.search("[$#@]",p):

continue

elif re.search("\s",p):

continue

else:

pass

value.append(p)

print ",".join(value)

#----------------------------------------#

#----------------------------------------#

Question 19

Level 3

Question:

You are required to write a program to sort the (name, age, height) tuples by ascending order where name is string, age and height are numbers. The tuples are input by console. The sort criteria is:

1: Sort based on name;

2: Then sort based on age;

3: Then sort by score.

The priority is that name > age > score.

If the following tuples are given as input to the program:

Tom,19,80

John,20,90

Jony,17,91

Jony,17,93

Json,21,85

Then, the output of the program should be:

[('John', '20', '90'), ('Jony', '17', '91'), ('Jony', '17', '93'), ('Json', '21', '85'), ('Tom', '19', '80')]

Hints:

In case of input data being supplied to the question, it should be assumed to be a console input.

We use itemgetter to enable multiple sort keys.

Solutions:

from operator import itemgetter, attrgetter

l = []

while True:

s = raw\_input()

if not s:

break

l.append(tuple(s.split(",")))

print sorted(l, key=itemgetter(0,1,2))

#----------------------------------------#

#----------------------------------------#

Question 20

Level 3

Question:

Define a class with a generator which can iterate the numbers, which are divisible by 7, between a given range 0 and n.

Hints:

Consider use yield

Solution:

def putNumbers(n):

i = 0

while i<n:

j=i

i=i+1

if j%7==0:

yield j

for i in reverse(100):

print i

#----------------------------------------#

#----------------------------------------#

Question 21

Level 3

Question£º

A robot moves in a plane starting from the original point (0,0). The robot can move toward UP, DOWN, LEFT and RIGHT with a given steps. The trace of robot movement is shown as the following:

UP 5

DOWN 3

LEFT 3

RIGHT 2

¡­

The numbers after the direction are steps. Please write a program to compute the distance from current position after a sequence of movement and original point. If the distance is a float, then just print the nearest integer.

Example:

If the following tuples are given as input to the program:

UP 5

DOWN 3

LEFT 3

RIGHT 2

Then, the output of the program should be:

2

Hints:

In case of input data being supplied to the question, it should be assumed to be a console input.

Solution:

import math

pos = [0,0]

while True:

s = raw\_input()

if not s:

break

movement = s.split(" ")

direction = movement[0]

steps = int(movement[1])

if direction=="UP":

pos[0]+=steps

elif direction=="DOWN":

pos[0]-=steps

elif direction=="LEFT":

pos[1]-=steps

elif direction=="RIGHT":

pos[1]+=steps

else:

pass

print int(round(math.sqrt(pos[1]\*\*2+pos[0]\*\*2)))

#----------------------------------------#

#----------------------------------------#

Question 22

Level 3

Question:

Write a program to compute the frequency of the words from the input. The output should output after sorting the key alphanumerically.

Suppose the following input is supplied to the program:

New to Python or choosing between Python 2 and Python 3? Read Python 2 or Python 3.

Then, the output should be:

2:2

3.:1

3?:1

New:1

Python:5

Read:1

and:1

between:1

choosing:1

or:2

to:1

Hints

In case of input data being supplied to the question, it should be assumed to be a console input.

Solution:

freq = {} # frequency of words in text

line = raw\_input()

for word in line.split():

freq[word] = freq.get(word,0)+1

words = freq.keys()

words.sort()

for w in words:

print "%s:%d" % (w,freq[w])

#----------------------------------------#

#----------------------------------------#

Question 23

level 1

Question:

Write a method which can calculate square value of number

Hints:

Using the \*\* operator

Solution:

def square(num):

return num \*\* 2

print square(2)

print square(3)

#----------------------------------------#

#----------------------------------------#

Question 24

Level 1

Question:

Python has many built-in functions, and if you do not know how to use it, you can read document online or find some books. But Python has a built-in document function for every built-in functions.

Please write a program to print some Python built-in functions documents, such as abs(), int(), raw\_input()

And add document for your own function

Hints:

The built-in document method is \_\_doc\_\_

Solution:

print abs.\_\_doc\_\_

print int.\_\_doc\_\_

print raw\_input.\_\_doc\_\_

def square(num):

'''Return the square value of the input number.

The input number must be integer.

'''

return num \*\* 2

print square(2)

print square.\_\_doc\_\_

#----------------------------------------#

#----------------------------------------#

Question 25

Level 1

Question:

Define a class, which have a class parameter and have a same instance parameter.

Hints:

Define a instance parameter, need add it in \_\_init\_\_ method

You can init a object with construct parameter or set the value later

Solution:

class Person:

# Define the class parameter "name"

name = "Person"

def \_\_init\_\_(self, name = None):

# self.name is the instance parameter

self.name = name

jeffrey = Person("Jeffrey")

print "%s name is %s" % (Person.name, jeffrey.name)

nico = Person()

nico.name = "Nico"

print "%s name is %s" % (Person.name, nico.name)

#----------------------------------------#

#----------------------------------------#

Question:

Define a function which can compute the sum of two numbers.

Hints:

Define a function with two numbers as arguments. You can compute the sum in the function and return the value.

Solution

def SumFunction(number1, number2):

return number1+number2

print SumFunction(1,2)

#----------------------------------------#

Question:

Define a function that can convert a integer into a string and print it in console.

Hints:

Use str() to convert a number to string.

Solution

def printValue(n):

print str(n)

printValue(3)

#----------------------------------------#

Question:

Define a function that can convert a integer into a string and print it in console.

Hints:

Use str() to convert a number to string.

Solution

def printValue(n):

print str(n)

printValue(3)

#----------------------------------------#

2.10

Question:

Define a function that can receive two integral numbers in string form and compute their sum and then print it in console.

Hints:

Use int() to convert a string to integer.

Solution

def printValue(s1,s2):

print int(s1)+int(s2)

printValue("3","4") #7

#----------------------------------------#

2.10

Question:

Define a function that can accept two strings as input and concatenate them and then print it in console.

Hints:

Use + to concatenate the strings

Solution

def printValue(s1,s2):

print s1+s2

printValue("3","4") #34

#----------------------------------------#

2.10

Question:

Define a function that can accept two strings as input and print the string with maximum length in console. If two strings have the same length, then the function should print al l strings line by line.

Hints:

Use len() function to get the length of a string

Solution

def printValue(s1,s2):

len1 = len(s1)

len2 = len(s2)

if len1>len2:

print s1

elif len2>len1:

print s2

else:

print s1

print s2

printValue("one","three")

#----------------------------------------#

2.10

Question:

Define a function that can accept an integer number as input and print the "It is an even number" if the number is even, otherwise print "It is an odd number".

Hints:

Use % operator to check if a number is even or odd.

Solution

def checkValue(n):

if n%2 == 0:

print "It is an even number"

else:

print "It is an odd number"

checkValue(7)

#----------------------------------------#

2.10

Question:

Define a function which can print a dictionary where the keys are numbers between 1 and 3 (both included) and the values are square of keys.

Hints:

Use dict[key]=value pattern to put entry into a dictionary.

Use \*\* operator to get power of a number.

Solution

def printDict():

d=dict()

d[1]=1

d[2]=2\*\*2

d[3]=3\*\*2

print d

printDict()

#----------------------------------------#

2.10

Question:

Define a function which can print a dictionary where the keys are numbers between 1 and 20 (both included) and the values are square of keys.

Hints:

Use dict[key]=value pattern to put entry into a dictionary.

Use \*\* operator to get power of a number.

Use range() for loops.

Solution

def printDict():

d=dict()

for i in range(1,21):

d[i]=i\*\*2

print d

printDict()

#----------------------------------------#

2.10

Question:

Define a function which can generate a dictionary where the keys are numbers between 1 and 20 (both included) and the values are square of keys. The function should just print the values only.

Hints:

Use dict[key]=value pattern to put entry into a dictionary.

Use \*\* operator to get power of a number.

Use range() for loops.

Use keys() to iterate keys in the dictionary. Also we can use item() to get key/value pairs.

Solution

def printDict():

d=dict()

for i in range(1,21):

d[i]=i\*\*2

for (k,v) in d.items():

print v

printDict()

#----------------------------------------#

2.10

Question:

Define a function which can generate a dictionary where the keys are numbers between 1 and 20 (both included) and the values are square of keys. The function should just print the keys only.

Hints:

Use dict[key]=value pattern to put entry into a dictionary.

Use \*\* operator to get power of a number.

Use range() for loops.

Use keys() to iterate keys in the dictionary. Also we can use item() to get key/value pairs.

Solution

def printDict():

d=dict()

for i in range(1,21):

d[i]=i\*\*2

for k in d.keys():

print k

printDict()

#----------------------------------------#

2.10

Question:

Define a function which can generate and print a list where the values are square of numbers between 1 and 20 (both included).

Hints:

Use \*\* operator to get power of a number.

Use range() for loops.

Use list.append() to add values into a list.

Solution

def printList():

li=list()

for i in range(1,21):

li.append(i\*\*2)

print li

printList()

#----------------------------------------#

2.10

Question:

Define a function which can generate a list where the values are square of numbers between 1 and 20 (both included). Then the function needs to print the first 5 elements in the list.

Hints:

Use \*\* operator to get power of a number.

Use range() for loops.

Use list.append() to add values into a list.

Use [n1:n2] to slice a list

Solution

def printList():

li=list()

for i in range(1,21):

li.append(i\*\*2)

print li[:5]

printList()

#----------------------------------------#

2.10

Question:

Define a function which can generate a list where the values are square of numbers between 1 and 20 (both included). Then the function needs to print the last 5 elements in the list.

Hints:

Use \*\* operator to get power of a number.

Use range() for loops.

Use list.append() to add values into a list.

Use [n1:n2] to slice a list

Solution

def printList():

li=list()

for i in range(1,21):

li.append(i\*\*2)

print li[-5:]

printList()

#----------------------------------------#

2.10

Question:

Define a function which can generate a list where the values are square of numbers between 1 and 20 (both included). Then the function needs to print all values except the first 5 elements in the list.

Hints:

Use \*\* operator to get power of a number.

Use range() for loops.

Use list.append() to add values into a list.

Use [n1:n2] to slice a list

Solution

def printList():

li=list()

for i in range(1,21):

li.append(i\*\*2)

print li[5:]

printList()

#----------------------------------------#

2.10

Question:

Define a function which can generate and print a tuple where the value are square of numbers between 1 and 20 (both included).

Hints:

Use \*\* operator to get power of a number.

Use range() for loops.

Use list.append() to add values into a list.

Use tuple() to get a tuple from a list.

Solution

def printTuple():

li=list()

for i in range(1,21):

li.append(i\*\*2)

print tuple(li)

printTuple()

#----------------------------------------#

2.10

Question:

With a given tuple (1,2,3,4,5,6,7,8,9,10), write a program to print the first half values in one line and the last half values in one line.

Hints:

Use [n1:n2] notation to get a slice from a tuple.

Solution

tp=(1,2,3,4,5,6,7,8,9,10)

tp1=tp[:5]

tp2=tp[5:]

print tp1

print tp2

#----------------------------------------#

2.10

Question:

Write a program to generate and print another tuple whose values are even numbers in the given tuple (1,2,3,4,5,6,7,8,9,10).

Hints:

Use "for" to iterate the tuple

Use tuple() to generate a tuple from a list.

Solution

tp=(1,2,3,4,5,6,7,8,9,10)

li=list()

for i in tp:

if tp[i]%2==0:

li.append(tp[i])

tp2=tuple(li)

print tp2

#----------------------------------------#

2.14

Question:

Write a program which accepts a string as input to print "Yes" if the string is "yes" or "YES" or "Yes", otherwise print "No".

Hints:

Use if statement to judge condition.

Solution

s= raw\_input()

if s=="yes" or s=="YES" or s=="Yes":

print "Yes"

else:

print "No"

#----------------------------------------#

3.4

Question:

Write a program which can filter even numbers in a list by using filter function. The list is: [1,2,3,4,5,6,7,8,9,10].

Hints:

Use filter() to filter some elements in a list.

Use lambda to define anonymous functions.

Solution

li = [1,2,3,4,5,6,7,8,9,10]

evenNumbers = filter(lambda x: x%2==0, li)

print evenNumbers

#----------------------------------------#

3.4

Question:

Write a program which can map() to make a list whose elements are square of elements in [1,2,3,4,5,6,7,8,9,10].

Hints:

Use map() to generate a list.

Use lambda to define anonymous functions.

Solution

li = [1,2,3,4,5,6,7,8,9,10]

squaredNumbers = map(lambda x: x\*\*2, li)

print squaredNumbers

#----------------------------------------#

3.5

Question:

Write a program which can map() and filter() to make a list whose elements are square of even number in [1,2,3,4,5,6,7,8,9,10].

Hints:

Use map() to generate a list.

Use filter() to filter elements of a list.

Use lambda to define anonymous functions.

Solution

li = [1,2,3,4,5,6,7,8,9,10]

evenNumbers = map(lambda x: x\*\*2, filter(lambda x: x%2==0, li))

print evenNumbers

#----------------------------------------#

3.5

Question:

Write a program which can filter() to make a list whose elements are even number between 1 and 20 (both included).

Hints:

Use filter() to filter elements of a list.

Use lambda to define anonymous functions.

Solution

evenNumbers = filter(lambda x: x%2==0, range(1,21))

print evenNumbers

#----------------------------------------#

3.5

Question:

Write a program which can map() to make a list whose elements are square of numbers between 1 and 20 (both included).

Hints:

Use map() to generate a list.

Use lambda to define anonymous functions.

Solution

squaredNumbers = map(lambda x: x\*\*2, range(1,21))

print squaredNumbers

#----------------------------------------#

7.2

Question:

Define a class named American which has a static method called printNationality.

Hints:

Use @staticmethod decorator to define class static method.

Solution

class American(object):

@staticmethod

def printNationality():

print "America"

anAmerican = American()

anAmerican.printNationality()

American.printNationality()

#----------------------------------------#

7.2

Question:

Define a class named American and its subclass NewYorker.

Hints:

Use class Subclass(ParentClass) to define a subclass.

Solution:

class American(object):

pass

class NewYorker(American):

pass

anAmerican = American()

aNewYorker = NewYorker()

print anAmerican

print aNewYorker

#----------------------------------------#

7.2

Question:

Define a class named Circle which can be constructed by a radius. The Circle class has a method which can compute the area.

Hints:

Use def methodName(self) to define a method.

Solution:

class Circle(object):

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

self.radius = r

def area(self):

return self.radius\*\*2\*3.14

aCircle = Circle(2)

print aCircle.area()

#----------------------------------------#

7.2

Define a class named Rectangle which can be constructed by a length and width. The Rectangle class has a method which can compute the area.

Hints:

Use def methodName(self) to define a method.

Solution:

class Rectangle(object):

def \_\_init\_\_(self, l, w):

self.length = l

self.width = w

def area(self):

return self.length\*self.width

aRectangle = Rectangle(2,10)

print aRectangle.area()

#----------------------------------------#

7.2

Define a class named Shape and its subclass Square. The Square class has an init function which takes a length as argument. Both classes have a area function which can print the area of the shape where Shape's area is 0 by default.

Hints:

To override a method in super class, we can define a method with the same name in the super class.

Solution:

class Shape(object):

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

pass

def area(self):

return 0

class Square(Shape):

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

Shape.\_\_init\_\_(self)

self.length = l

def area(self):

return self.length\*self.length

aSquare= Square(3)

print aSquare.area()

#----------------------------------------#

Please raise a RuntimeError exception.

Hints:

Use raise() to raise an exception.

Solution:

raise RuntimeError('something wrong')

#----------------------------------------#

Write a function to compute 5/0 and use try/except to catch the exceptions.

Hints:

Use try/except to catch exceptions.

Solution:

def throws():

return 5/0

try:

throws()

except ZeroDivisionError:

print "division by zero!"

except Exception, err:

print 'Caught an exception'

finally:

print 'In finally block for cleanup'

#----------------------------------------#

Define a custom exception class which takes a string message as attribute.

Hints:

To define a custom exception, we need to define a class inherited from Exception.

Solution:

class MyError(Exception):

"""My own exception class

Attributes:

msg -- explanation of the error

"""

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

self.msg = msg

error = MyError("something wrong")

#----------------------------------------#

Question:

Assuming that we have some email addresses in the "username@companyname.com" format, please write program to print the user name of a given email address. Both user names and company names are composed of letters only.

Example:

If the following email address is given as input to the program:

john@google.com

Then, the output of the program should be:

john

In case of input data being supplied to the question, it should be assumed to be a console input.

Hints:

Use \w to match letters.

Solution:

import re

emailAddress = raw\_input()

pat2 = "(\w+)@((\w+\.)+(com))"

r2 = re.match(pat2,emailAddress)

print r2.group(1)

#----------------------------------------#

Question:

Assuming that we have some email addresses in the "username@companyname.com" format, please write program to print the company name of a given email address. Both user names and company names are composed of letters only.

Example:

If the following email address is given as input to the program:

john@google.com

Then, the output of the program should be:

google

In case of input data being supplied to the question, it should be assumed to be a console input.

Hints:

Use \w to match letters.

Solution:

import re

emailAddress = raw\_input()

pat2 = "(\w+)@(\w+)\.(com)"

r2 = re.match(pat2,emailAddress)

print r2.group(2)

#----------------------------------------#

Question:

Write a program which accepts a sequence of words separated by whitespace as input to print the words composed of digits only.

Example:

If the following words is given as input to the program:

2 cats and 3 dogs.

Then, the output of the program should be:

['2', '3']

In case of input data being supplied to the question, it should be assumed to be a console input.

Hints:

Use re.findall() to find all substring using regex.

Solution:

import re

s = raw\_input()

print re.findall("\d+",s)

#----------------------------------------#

Question:

Print a unicode string "hello world".

Hints:

Use u'strings' format to define unicode string.

Solution:

unicodeString = u"hello world!"

print unicodeString

#----------------------------------------#

Write a program to read an ASCII string and to convert it to a unicode string encoded by utf-8.

Hints:

Use unicode() function to convert.

Solution:

s = raw\_input()

u = unicode( s ,"utf-8")

print u

#----------------------------------------#

Question:

Write a special comment to indicate a Python source code file is in unicode.

Hints:

Solution:

# -\*- coding: utf-8 -\*-

#----------------------------------------#

Question:

Write a program to compute 1/2+2/3+3/4+...+n/n+1 with a given n input by console (n>0).

Example:

If the following n is given as input to the program:

5

Then, the output of the program should be:

3.55

In case of input data being supplied to the question, it should be assumed to be a console input.

Hints:

Use float() to convert an integer to a float

Solution:

n=int(raw\_input())

sum=0.0

for i in range(1,n+1):

sum += float(float(i)/(i+1))

print sum

#----------------------------------------#

Question:

Write a program to compute:

f(n)=f(n-1)+100 when n>0

and f(0)=1

with a given n input by console (n>0).

Example:

If the following n is given as input to the program:

5

Then, the output of the program should be:

500

In case of input data being supplied to the question, it should be assumed to be a console input.

Hints:

We can define recursive function in Python.

Solution:

def f(n):

if n==0:

return 0

else:

return f(n-1)+100

n=int(raw\_input())

print f(n)

#----------------------------------------#

Question:

The Fibonacci Sequence is computed based on the following formula:

f(n)=0 if n=0

f(n)=1 if n=1

f(n)=f(n-1)+f(n-2) if n>1

Please write a program to compute the value of f(n) with a given n input by console.

Example:

If the following n is given as input to the program:

7

Then, the output of the program should be:

13

In case of input data being supplied to the question, it should be assumed to be a console input.

Hints:

We can define recursive function in Python.

Solution:

def f(n):

if n == 0: return 0

elif n == 1: return 1

else: return f(n-1)+f(n-2)

n=int(raw\_input())

print f(n)

#----------------------------------------#

#----------------------------------------#

Question:

The Fibonacci Sequence is computed based on the following formula:

f(n)=0 if n=0

f(n)=1 if n=1

f(n)=f(n-1)+f(n-2) if n>1

Please write a program using list comprehension to print the Fibonacci Sequence in comma separated form with a given n input by console.

Example:

If the following n is given as input to the program:

7

Then, the output of the program should be:

0,1,1,2,3,5,8,13

Hints:

We can define recursive function in Python.

Use list comprehension to generate a list from an existing list.

Use string.join() to join a list of strings.

In case of input data being supplied to the question, it should be assumed to be a console input.

Solution:

def f(n):

if n == 0: return 0

elif n == 1: return 1

else: return f(n-1)+f(n-2)

n=int(raw\_input())

values = [str(f(x)) for x in range(0, n+1)]

print ",".join(values)

#----------------------------------------#

Question:

Please write a program using generator to print the even numbers between 0 and n in comma separated form while n is input by console.

Example:

If the following n is given as input to the program:

10

Then, the output of the program should be:

0,2,4,6,8,10

Hints:

Use yield to produce the next value in generator.

In case of input data being supplied to the question, it should be assumed to be a console input.

Solution:

def EvenGenerator(n):

i=0

while i<=n:

if i%2==0:

yield i

i+=1

n=int(raw\_input())

values = []

for i in EvenGenerator(n):

values.append(str(i))

print ",".join(values)

#----------------------------------------#

Question:

Please write a program using generator to print the numbers which can be divisible by 5 and 7 between 0 and n in comma separated form while n is input by console.

Example:

If the following n is given as input to the program:

100

Then, the output of the program should be:

0,35,70

Hints:

Use yield to produce the next value in generator.

In case of input data being supplied to the question, it should be assumed to be a console input.

Solution:

def NumGenerator(n):

for i in range(n+1):

if i%5==0 and i%7==0:

yield i

n=int(raw\_input())

values = []

for i in NumGenerator(n):

values.append(str(i))

print ",".join(values)

#----------------------------------------#

Question:

Please write assert statements to verify that every number in the list [2,4,6,8] is even.

Hints:

Use "assert expression" to make assertion.

Solution:

li = [2,4,6,8]

for i in li:

assert i%2==0

#----------------------------------------#

Question:

Please write a program which accepts basic mathematic expression from console and print the evaluation result.

Example:

If the following string is given as input to the program:

35+3

Then, the output of the program should be:

38

Hints:

Use eval() to evaluate an expression.

Solution:

expression = raw\_input()

print eval(expression)

#----------------------------------------#

Question:

Please write a binary search function which searches an item in a sorted list. The function should return the index of element to be searched in the list.

Hints:

Use if/elif to deal with conditions.

Solution:

import math

def bin\_search(li, element):

bottom = 0

top = len(li)-1

index = -1

while top>=bottom and index==-1:

mid = int(math.floor((top+bottom)/2.0))

if li[mid]==element:

index = mid

elif li[mid]>element:

top = mid-1

else:

bottom = mid+1

return index

li=[2,5,7,9,11,17,222]

print bin\_search(li,11)

print bin\_search(li,12)

#----------------------------------------#

Question:

Please write a binary search function which searches an item in a sorted list. The function should return the index of element to be searched in the list.

Hints:

Use if/elif to deal with conditions.

Solution:

import math

def bin\_search(li, element):

bottom = 0

top = len(li)-1

index = -1

while top>=bottom and index==-1:

mid = int(math.floor((top+bottom)/2.0))

if li[mid]==element:

index = mid

elif li[mid]>element:

top = mid-1

else:

bottom = mid+1

return index

li=[2,5,7,9,11,17,222]

print bin\_search(li,11)

print bin\_search(li,12)

#----------------------------------------#

Question:

Please generate a random float where the value is between 10 and 100 using Python math module.

Hints:

Use random.random() to generate a random float in [0,1].

Solution:

import random

print random.random()\*100

#----------------------------------------#

Question:

Please generate a random float where the value is between 5 and 95 using Python math module.

Hints:

Use random.random() to generate a random float in [0,1].

Solution:

import random

print random.random()\*100-5

#----------------------------------------#

Question:

Please write a program to output a random even number between 0 and 10 inclusive using random module and list comprehension.

Hints:

Use random.choice() to a random element from a list.

Solution:

import random

print random.choice([i for i in range(11) if i%2==0])

#----------------------------------------#

Question:

Please write a program to output a random number, which is divisible by 5 and 7, between 0 and 10 inclusive using random module and list comprehension.

Hints:

Use random.choice() to a random element from a list.

Solution:

import random

print random.choice([i for i in range(201) if i%5==0 and i%7==0])

#----------------------------------------#

Question:

Please write a program to generate a list with 5 random numbers between 100 and 200 inclusive.

Hints:

Use random.sample() to generate a list of random values.

Solution:

import random

print random.sample(range(100), 5)

#----------------------------------------#

Question:

Please write a program to randomly generate a list with 5 even numbers between 100 and 200 inclusive.

Hints:

Use random.sample() to generate a list of random values.

Solution:

import random

print random.sample([i for i in range(100,201) if i%2==0], 5)

#----------------------------------------#

Question:

Please write a program to randomly generate a list with 5 numbers, which are divisible by 5 and 7 , between 1 and 1000 inclusive.

Hints:

Use random.sample() to generate a list of random values.

Solution:

import random

print random.sample([i for i in range(1,1001) if i%5==0 and i%7==0], 5)

#----------------------------------------#

Question:

Please write a program to randomly print a integer number between 7 and 15 inclusive.

Hints:

Use random.randrange() to a random integer in a given range.

Solution:

import random

print random.randrange(7,16)

#----------------------------------------#

Question:

Please write a program to compress and decompress the string "hello world!hello world!hello world!hello world!".

Hints:

Use zlib.compress() and zlib.decompress() to compress and decompress a string.

Solution:

import zlib

s = 'hello world!hello world!hello world!hello world!'

t = zlib.compress(s)

print t

print zlib.decompress(t)

#----------------------------------------#

Question:

Please write a program to print the running time of execution of "1+1" for 100 times.

Hints:

Use timeit() function to measure the running time.

Solution:

from timeit import Timer

t = Timer("for i in range(100):1+1")

print t.timeit()

#----------------------------------------#

Question:

Please write a program to shuffle and print the list [3,6,7,8].

Hints:

Use shuffle() function to shuffle a list.

Solution:

from random import shuffle

li = [3,6,7,8]

shuffle(li)

print li

#----------------------------------------#

Question:

Please write a program to shuffle and print the list [3,6,7,8].

Hints:

Use shuffle() function to shuffle a list.

Solution:

from random import shuffle

li = [3,6,7,8]

shuffle(li)

print li

#----------------------------------------#

Question:

Please write a program to generate all sentences where subject is in ["I", "You"] and verb is in ["Play", "Love"] and the object is in ["Hockey","Football"].

Hints:

Use list[index] notation to get a element from a list.

Solution:

subjects=["I", "You"]

verbs=["Play", "Love"]

objects=["Hockey","Football"]

for i in range(len(subjects)):

for j in range(len(verbs)):

for k in range(len(objects)):

sentence = "%s %s %s." % (subjects[i], verbs[j], objects[k])

print sentence

#----------------------------------------#

Please write a program to print the list after removing delete even numbers in [5,6,77,45,22,12,24].

Hints:

Use list comprehension to delete a bunch of element from a list.

Solution:

li = [5,6,77,45,22,12,24]

li = [x for x in li if x%2!=0]

print li

#----------------------------------------#

Question:

By using list comprehension, please write a program to print the list after removing delete numbers which are divisible by 5 and 7 in [12,24,35,70,88,120,155].

Hints:

Use list comprehension to delete a bunch of element from a list.

Solution:

li = [12,24,35,70,88,120,155]

li = [x for x in li if x%5!=0 and x%7!=0]

print li

#----------------------------------------#

Question:

By using list comprehension, please write a program to print the list after removing the 0th, 2nd, 4th,6th numbers in [12,24,35,70,88,120,155].

Hints:

Use list comprehension to delete a bunch of element from a list.

Use enumerate() to get (index, value) tuple.

Solution:

li = [12,24,35,70,88,120,155]

li = [x for (i,x) in enumerate(li) if i%2!=0]

print li

#----------------------------------------#

Question:

By using list comprehension, please write a program generate a 3\*5\*8 3D array whose each element is 0.

Hints:

Use list comprehension to make an array.

Solution:

array = [[ [0 for col in range(8)] for col in range(5)] for row in range(3)]

print array

#----------------------------------------#

Question:

By using list comprehension, please write a program to print the list after removing the 0th,4th,5th numbers in [12,24,35,70,88,120,155].

Hints:

Use list comprehension to delete a bunch of element from a list.

Use enumerate() to get (index, value) tuple.

Solution:

li = [12,24,35,70,88,120,155]

li = [x for (i,x) in enumerate(li) if i not in (0,4,5)]

print li

#----------------------------------------#

Question:

By using list comprehension, please write a program to print the list after removing the value 24 in [12,24,35,24,88,120,155].

Hints:

Use list's remove method to delete a value.

Solution:

li = [12,24,35,24,88,120,155]

li = [x for x in li if x!=24]

print li

#----------------------------------------#

Question:

With two given lists [1,3,6,78,35,55] and [12,24,35,24,88,120,155], write a program to make a list whose elements are intersection of the above given lists.

Hints:

Use set() and "&=" to do set intersection operation.

Solution:

set1=set([1,3,6,78,35,55])

set2=set([12,24,35,24,88,120,155])

set1 &= set2

li=list(set1)

print li

#----------------------------------------#

With a given list [12,24,35,24,88,120,155,88,120,155], write a program to print this list after removing all duplicate values with original order reserved.

Hints:

Use set() to store a number of values without duplicate.

Solution:

def removeDuplicate( li ):

newli=[]

seen = set()

for item in li:

if item not in seen:

seen.add( item )

newli.append(item)

return newli

li=[12,24,35,24,88,120,155,88,120,155]

print removeDuplicate(li)

#----------------------------------------#

Question:

Define a class Person and its two child classes: Male and Female. All classes have a method "getGender" which can print "Male" for Male class and "Female" for Female class.

Hints:

Use Subclass(Parentclass) to define a child class.

Solution:

class Person(object):

def getGender( self ):

return "Unknown"

class Male( Person ):

def getGender( self ):

return "Male"

class Female( Person ):

def getGender( self ):

return "Female"

aMale = Male()

aFemale= Female()

print aMale.getGender()

print aFemale.getGender()

#----------------------------------------#

Question:

Please write a program which count and print the numbers of each character in a string input by console.

Example:

If the following string is given as input to the program:

abcdefgabc

Then, the output of the program should be:

a,2

c,2

b,2

e,1

d,1

g,1

f,1

Hints:

Use dict to store key/value pairs.

Use dict.get() method to lookup a key with default value.

Solution:

dic = {}

s=raw\_input()

for s in s:

dic[s] = dic.get(s,0)+1

print '\n'.join(['%s,%s' % (k, v) for k, v in dic.items()])

#----------------------------------------#

Question:

Please write a program which accepts a string from console and print it in reverse order.

Example:

If the following string is given as input to the program:

rise to vote sir

Then, the output of the program should be:

ris etov ot esir

Hints:

Use list[::-1] to iterate a list in a reverse order.

Solution:

s=raw\_input()

s = s[::-1]

print s

#----------------------------------------#

Question:

Please write a program which accepts a string from console and print the characters that have even indexes.

Example:

If the following string is given as input to the program:

H1e2l3l4o5w6o7r8l9d

Then, the output of the program should be:

Helloworld

Hints:

Use list[::2] to iterate a list by step 2.

Solution:

s=raw\_input()

s = s[::2]

print s

#----------------------------------------#

Question:

Please write a program which prints all permutations of [1,2,3]

Hints:

Use itertools.permutations() to get permutations of list.

Solution:

import itertools

print list(itertools.permutations([1,2,3]))

#----------------------------------------#

Question:

Write a program to solve a classic ancient Chinese puzzle:

We count 35 heads and 94 legs among the chickens and rabbits in a farm. How many rabbits and how many chickens do we have?

Hint:

Use for loop to iterate all possible solutions.

Solution:

def solve(numheads,numlegs):

ns='No solutions!'

for i in range(numheads+1):

j=numheads-i

if 2\*i+4\*j==numlegs:

return i,j

return ns,ns

numheads=35

numlegs=94

solutions=solve(numheads,numlegs)

print solutions

#----------------------------------------#

FAQ's:

|  |
| --- |
| [Python Program to Print Hello world!](https://www.programiz.com/python-programming/examples/hello-world) |
| [Python Program to Add Two Numbers](https://www.programiz.com/python-programming/examples/add-number) |
| [Python Program to Find the Square Root](https://www.programiz.com/python-programming/examples/square-root) |
| [Python Program to Calculate the Area of a Triangle](https://www.programiz.com/python-programming/examples/area-triangle) |
| [Python Program to Solve Quadratic Equation](https://www.programiz.com/python-programming/examples/quadratic-roots) |
| [Python Program to Swap Two Variables](https://www.programiz.com/python-programming/examples/swap-variables) |
| [Python Program to Generate a Random Number](https://www.programiz.com/python-programming/examples/random-number) |
| [Python Program to Convert Kilometers to Miles](https://www.programiz.com/python-programming/examples/km-mile) |
| [Python Program to Convert Celsius To Fahrenheit](https://www.programiz.com/python-programming/examples/celsius-fahrenheit) |
| [Python Program to Check if a Number is Positive, Negative or 0](https://www.programiz.com/python-programming/examples/positive-negative-zero) |
| [Python Program to Check if a Number is Odd or Even](https://www.programiz.com/python-programming/examples/odd-even) |
| [Python Program to Check Leap Year](https://www.programiz.com/python-programming/examples/leap-year) |
| [Python Program to Find the Largest Among Three Numbers](https://www.programiz.com/python-programming/examples/largest-number-three) |
| [Python Program to Check Prime Number](https://www.programiz.com/python-programming/examples/prime-number) |
| [Python Program to Print all Prime Numbers in an Interval](https://www.programiz.com/python-programming/examples/prime-number-intervals) |
| [Python Program to Find the Factorial of a Number](https://www.programiz.com/python-programming/examples/factorial) |
| [Python Program to Display the multiplication Table](https://www.programiz.com/python-programming/examples/multiplication-table) |
| [Python Program to Print the Fibonacci sequence](https://www.programiz.com/python-programming/examples/fibonacci-sequence) |
| [Python Program to Check Armstrong Number](https://www.programiz.com/python-programming/examples/armstrong-number) |
| [Python Program to Find Armstrong Number in an Interval](https://www.programiz.com/python-programming/examples/armstrong-interval) |
| [Python Program to Find the Sum of Natural Numbers](https://www.programiz.com/python-programming/examples/sum-natural-number) |
| [Python Program To Display Powers of 2 Using Anonymous Function](https://www.programiz.com/python-programming/examples/power-anonymous) |
| [Python Program to Find Numbers Divisible by Another Number](https://www.programiz.com/python-programming/examples/number-divisible) |
| [Python Program to Convert Decimal to Binary, Octal and Hexadecimal](https://www.programiz.com/python-programming/examples/conversion-binary-octal-hexadecimal) |
| [Python Program to Find ASCII Value of Character](https://www.programiz.com/python-programming/examples/ascii-character) |
| [Python Program to Find HCF or GCD](https://www.programiz.com/python-programming/examples/hcf) |
| [Python Program to Find LCM](https://www.programiz.com/python-programming/examples/lcm) |
| [Python Program to Find Factors of Number](https://www.programiz.com/python-programming/examples/factor-number) |
| [Python Program to Make a Simple Calculator](https://www.programiz.com/python-programming/examples/calculator) |
| [Python Program to Shuffle Deck of Cards](https://www.programiz.com/python-programming/examples/shuffle-card) |
| [Python Program to Display Calendar](https://www.programiz.com/python-programming/examples/display-calendar) |
| [Python Program to Display Fibonacci Sequence Using Recursion](https://www.programiz.com/python-programming/examples/fibonacci-recursion) |
| [Python Program to Find Sum of Natural Numbers Using Recursion](https://www.programiz.com/python-programming/examples/natural-number-recursion) |
| [Python Program to Find Factorial of Number Using Recursion](https://www.programiz.com/python-programming/examples/factorial-recursion) |
| [Python Program to Convert Decimal to Binary Using Recursion](https://www.programiz.com/python-programming/examples/decimal-binary-recursion) |
| [Python Program to Add Two Matrices](https://www.programiz.com/python-programming/examples/add-matrix) |
| [Python Program to Transpose a Matrix](https://www.programiz.com/python-programming/examples/transpose-matrix) |
| [Python Program to Multiply Two Matrices](https://www.programiz.com/python-programming/examples/multiply-matrix) |
| [Python Program to Check Whether a String is Palindrome or Not](https://www.programiz.com/python-programming/examples/palindrome) |
| [Python Program to Remove Punctuations From a String](https://www.programiz.com/python-programming/examples/remove-punctuation) |
| [Python Program to Sort Words in Alphabetic Order](https://www.programiz.com/python-programming/examples/alphabetical-order) |
| [Python Program to Illustrate Different Set Operations](https://www.programiz.com/python-programming/examples/set-operation) |
| [Python Program to Count the Number of Each Vowel](https://www.programiz.com/python-programming/examples/count-vowel) |

Interview Questions:

1. Write a Python program to search the particular word present in given String or not?
2. Write a Python Program to replace some words with given words with in String?
3. What is import and use of it?
4. DIfference between Python and C, features of Python
5. Explain Functions and dictionary of Python. Take a dictionary of 100 pairs, print keys, values, pairs
6. What Is output of this expression dict(itertools. izip\_longest('a','b') (range(100) ) ) If it Is possible, If it Is not why . If possible Explain How and Is there any data loss?.
7. How can we read the data from csv file in Python
8. List contain [1,13,7,5,9], find which values have the addition 14 and find index no of that values
9. Write a Python program to print How many zeros and one's in a file , the file contians address and data, Example:data. Txt 0000:1, 0001:2, 0003:3.
10. Write a Python code to read a particular string and show what that string contains? Example: Data. txt: Filename:string, u need to show What filename contains.
11. Having list with some number of elements and try to add the elements at the starting without using attributes and loops?
12. List containing 1's and 0's and try to find the count of 1's without travelling into the list.
13. Why not Python why c++?
14. What is lambda function and return?
15. Difference between dictionary, list and tuple.
16. What are the basic data types in Python?
17. Explain about classes? What is the use of “self” in classes? What is the syntax of class and methods?
18. What “yield” will do? What is the difference between yield and for loop?
19. How to slice the string with a delimiter?
20. What is list comprehension? What is dictionary comprehension? Give one example on each?
21. What is call by value and call by reference in Python?
22. What is filter?
23. How to get the multiplication of all elements in the list?
24. Explain multi-threading in Python. How threads will work in Python? How to create thread and how to give Priority to the created thread? Explain, CR region in Multi-threading
25. Write a program to accept two numbers with one thread adding two numbers and other thread will substact two numbers, print the each thread's response in the common text file.   Should be this way:  result from addtion:     result from substraction
26. What is generator?
27. Write logic if for every open brace there is closed brace then return True else false,ex: f({{}{}{}}) -True,f(}{) -False,f({{{{}}}) -False
28. Write a Python program for append a string through commond line arguments.
29. How C libraries can be included in Python?
30. How to define default arguments in a function?
31. How to print last six characters in given string (Python)?
32. How to print last six elements in lists?
33. How remove duplicate elements in list ?
34. What is regular expression in Python? What is the use of re?
35. How to find the data in between two strings from a file using regular expressions?
36. Write one Python program using functions and explain?
37. Why we use tabbing in functions Python?
38. Is there any data types specification for variables in Python?
39. Write a Python progrom define a function in it and explain?
40. Write the function to print all the sub-directories and files in a given path. Import appropriate modules to interact with the OS.
41. List some ways to get some Python code to run in a parallel way.
42. Describe Python's garbage collection mechanism in brief.
43. Write a program which takes 2 digits, X,Y as input and generates a 2-dimensional array. The element value in the i-th row and j-th column of the array should be i\*j.Note: i=0,1.., X-1; j=0,1,¡­Y-1.
44. Given an array of integers, write a program to find the largest number possible from a concatenation of the integers in the array.  Note: Each number must be used exactly once.  An input array:[1,2,3,4], returns 4,3,2,1
45. Divide a number by 3 without using +, -, \*, / or % operators. The number may be signed or unsigned. Note: Use bitwise operations only
46. Given an array of integers, write a program to find the largest number possible from a concatenation of the integers in the array.  Note: Each number must be used exactly once.  An input array:[1,2,3,4], returns 4,3,2,1
47. Divide a number by 3 without using +, -, \*, / or % operators. The number may be signed or unsigned. Note: Use bitwise operations only
48. Write a program which will find all such numbers which are divisible by 7 but are not a multiple of 5,between two numbers provided by user. The numbers obtained should be printed in a comma-separated sequence on a single line.
49. Write a program which can compute the factorial of a given numbers.The results should be printed in a comma-separated sequence on a single line.
50. Suppose the following input is supplied to the program: 8 Then, the output should be: 40320;With a given integral number n, write a program to generate a dictionary that contains (i, i\*i) such that is an integral number between 1 and n (both included). and then the program should print the dictionary.
51. Suppose the following input is supplied to the program: 8 Then, the output should be:1.       {1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64}  Hints: In case of input data being supplied to the question, it should be assumed to be a console input.
52. Write a program which accepts a sequence of comma-separated numbers from console and generate a list and a tuple which contains every number.
53. Suppose the following input is supplied to the program:  34,67,55,33,12,98 .. Then, the output should be:  ['34', '67', '55', '33', '12', '98']  ('34', '67', '55', '33', '12', '98')
54. Define a class which has at least two methods: getString: to get a string from console input and printString: to print the string in upper case. Hint: Use \_\_init\_\_ method to construct some parameters
55. Write a program which takes 2 digits, X,Y as input and generates a 2-dimensional array. The element value in the i-th row and j-th column of the array should be i\*j. Note: i=0,1.., X-1; j=0,1,¡­Y-1.  Example  Suppose the following inputs are given to the program: 3,5 Then, the output of the program should be: [[0, 0, 0, 0, 0], [0, 1, 2, 3, 4], [0, 2, 4, 6, 8]]
56. Write a program that accepts sequence of lines as input and prints the lines after making all characters in the sentence capitalized.
57. Assume a 4x4 excel data as input and write a histogram chart for the same. Please assume the data missing. Emphasis Is on the histogram python code
58. Write a program that returns a list that contains only the elements that are common between the lists (without duplicates).
59. What are the different methods, Python provides for copying an object? Explain
60. How memory is managed in Python?
61. Explain how can you make a Python Script executable on Unix?
62. What are the built-in type does Python provides?
63. Using SQL create tables for 1. Open positions with entries(skills and number of positions) 2. Engineers Skill with entries ( Skill list, availability)
64. Write a Python scipt to establish sql query which will query the open positions, if open, get the matching skills from engineers and if available present the engineers name
65. John and Sally should be given bonus on salaries based on their sales, x%. Please write a python code using oops class, callback functions, registering and executing the call back to print the each salary + bonus.
66. Explain about Socket Programming. What is Socket?
67. Explain OSI Layers. What is TCP/IP layer? Write TCP block diagram.
68. Write a simple HTTP Client program in Python
69. How you can transfer whatsapp from 1 device to another explain using TCP/IP?
70. Write a Python code for creating server socket?
71. What are dIfferent types of topologies and Explain about bus, ring, star, mesh topologies?
72. How do you know which ports are opened in your machine.
73. What Is Transport stream? Explain about TCP/IP and also TCP Header. Why do we have both MAC address and IP address?
74. Differences between TCP and UDP
75. Explain Checksum and Cyclic Redundancy Check
76. Write in detail in Tabular format, differences between deep and shallow copies with examples