**Dear Learner,**

**In this module, you will get to learn about basics of Python programming.**

* **Request you to go through the course and complete it.**
* **After completing this course , you will learn about basic programming constructs such as variables,data types -Numbers,Lists,Tuples,Sets,Strings Dictionary,Arrays  ,Operators and control statements**
* **Finally , you will be able to write programs in Python and execute them.**
* **After going through the  contents provided, request you to take the quiz to understand your learning improvements.**
* **It is mandatory to complete the practice assignments provided in the Python\_Hands On course to proceed for the next module.**

**.1. Python Overview**

Python is a general-purpose, interpreted, high-level programming language which is widely used nowadays .It is an open source language which was developed by Guido Van Rossum in the late 1980s .Python Software Foundation(PSF), a non-profit organization, holds the intellectual property rights of Python.Python was released in 1991 at Centrum Wiskunde & Informatica (CWI) in the Netherlands as a successor to the ABC language. Rossum named this language after a popular comedy show called 'Monty Python's Flying Circus' (and not after Python-the snake).

In the last few years, the popularity of python has increased immensely due to its wide apllications. According to most of the tech surveys, Python is in the top ten Most Popular Technologies in 2019.

Official Web Site: [https://www.python.org](https://www.python.org/)

**1.1. Python Version History**

Currently, PSF supports two versions, Python 2.x & Python 3.x. Python 2.0 was released in October 2000 and includes a large number of features. PSF continues to support version Python 2 because a large body of existing code could not be forward ported to Python 3. So, they will support Python 2 until 2020.

Python 3.0 was released on December 3rd, 2008. It was designed to rectify certain flaws in earlier version. This version is not completely backward-compatible with previous versions. However, many of its major features have since been back-ported to the Python 2.6.x and 2.7.x version series. Releases of Python 3 include 2 to 3 utilities to facilitate the automation of translation of Python 2 code to Python 3.

**1.2. Python Features**

* Python is an interpreter-based language, which allows execution of one instruction at a time.
* Extensive basic data types are supported e.g. numbers (floating point, complex, and unlimited-length long integers), strings (both ASCII and Unicode), lists, and dictionaries.
* Variables can be strongly typed as well as dynamic typed.
* Supports object-oriented programming concepts such as class, inheritance, objects, module, namespace etc.
* Cleaner exception handling support.
* Supports automatic memory management.

**1.3. Python Advantages**

* Python provides enhanced readability. For that purpose, uniform indents are used to delimit blocks of statements instead of curly brackets, like in many languages such as C, C++ and Java.
* Python is free and distributed as open-source software. A large programming community is actively involved in the development and support of Python libraries for various applications such as web frameworks, mathematical computing and data science.
* Python is a cross-platform language. It works equally on different OS platforms like Windows, Linux, Mac OSX etc. Hence Python applications can be easily ported across OS platforms.
* Python supports multiple programming paradigms including imperative, procedural, object-oriented and functional programming styles.
* Python is an extensible language. Additional functionality (other than what is provided in the core language) can be made available through modules and packages written in other languages (C, C++, Java etc)
* A standard DB-API for database connectivity has been defined in Python. It can be enabled using any data source (Oracle, MySQL, SQLite etc.) as a backend to the Python program for storage, retrieval and processing of data.
* Standard distribution of Python contains the Tkinter GUI toolkit, which is the implementation of popular GUI library called Tcl/Tk. An attractive GUI can be constructed using Tkinter. Many other GUI libraries like Qt, GTK, WxWidgets etc. are also ported to Python.
* Python can be integrated with other popular programming technologies like C, C++, Java, ActiveX and CORBA.

**1.4. Python Applications**

Even though Python started as a general-purpose programming language with no particular application as its focus, over last few years it has emerged as the language of choice for developers in some application areas. Some important applications of Python are summarized below:

**1.4.1. Data Science**

Python gained a lot of popularity recently because of its Data science libraries. Huge amount of data is being generated today by web applications, mobile applications and other devices. Companies need business insights from this data. To analayze this data, the libraries of Python comes quite handy.

Today Python has become the language of choice for data scientists. Python libraries like NumPy, Pandas and Matplotlib are extensively used in the process of data analysis, including the collection, processing and cleansing of data sets, applying mathematical algorithms and generating visualizations for the benefit of users. Commercial and community Python distributions by third-parties such as Anaconda and ActiveState provide all the essential libraries required for data science.

**1.4.2. Machine Learning**

This is another key application area of Python. Python libraries such as Scikit-learn, Tensorflow and NLTK are widely used for the prediction of trends like customer satisfaction, projected values of stocks etc. Some of the real-world applications of machine learning include medical diagnosis,sales prediction, feedback analysis etc.

**1.4.3. Web Development**

This is another application area in which Python is becoming very popular. Web application framework libraries like django, Pyramid, Flask etc. make it very easy to develop and deploy simple as well as complex web applications. These frameworks are used extensively by various IT companies. Dropbox for example uses django as a backend to store and, synchronize local folders.

Most of the web servers today are compatible with WSGI (Web Server Gateway Interface) - a specification for the universal interface between Python web frameworks and web servers. All leading web servers such as Apache, IIS, Nginxetc. can now host Python web applications. Google's App Engine hosts web applications built with almost all Python web frameworks.

**1.4.4. Image Processing**

The OpenCV library is commonly used for face detection and gesture recognition. OpenCV is a C++ library, but has been ported to Python. Because of the rapid development of this feature, Python is a very popular choice from image processing.

**1.4.5. Game Development**

Python is a popular choice for game developers. The PyGame library is extensively used for building games for desktop as well as for mobile platforms. It includes a huge set of computer graphics and sound libraries. PyGame is a Python wrapper for the SDL library. SDL is a cross-platform library for accessing computer multimedia hardware components (sound, video, input, etc.). SDL is an extremely powerful tool for building all kinds of things, but it’s written in C, and hence in Python, pygame is used.

**1.4.6. Embedded Systems and IoT**

Another important area of Python application is in embedded systems. Raspberry Pi is a very popular yet a low-cost single-board computer. It is being extensively used in automation products, robotics, IoT, and kiosk applications. Popular microcontrollers like Arduino are used in many IoT products and are being programmed with Python. A lightweight version of Python called Micropython has been developed especially for microcontrollers. A special Micropython-compatible controller called PyBoard has also been developed.

**1.4.7. App development**

Python can also be used to develop Android apps. Python's Kivy library has all the functionalities required for a mobile application.Kivy is an open-source Python library for developing cross-platform GUI applications. It allows you to write pure-Python graphical applications that run on the main desktop platforms (Windows, Linux, and macOS) and on iOS & Android.

**1.4.8. Automated Jobs**

Python is extremely useful and widely used for automating CRON (Command Run ON) jobs. Certain tasks like backups, defined in Python scripts can be scheduled to be invoked automatically by the operating system scheduler to be executed at predefined times.

**1.4.9. Rapid Development Tool**

Standard distribution of Python as developed by Rossum and maintained by Python Software Foundation is called Cpython which is a reference implementation. Its alternative implementations -Jython the JRE implementation of Python and IronPython - the .NET implementation, interact seamlessly with Java and C#, respectively. For example Jython can use all Java libraries such as Swing etc. So the development time can be minimized by using simpler Python syntaxes and Java libraries for prototyping the software product.

**1.5. Reference Links & Videos**

1. <http://tdc-www.harvard.edu/Python.pdf>  
2.<http://www.seas.upenn.edu/~cis391/Lectures/python-tutorial.pdf>

**Python Programming Basics**

**1.1 Basic Syntax**

* Python syntax is highly readable.
* Statements in Python typically end with a new line.
* (\) is used to denote line continuation.
* Python uses indentation to indicate a block of code and gives error if indentation is skipped.
* All the continuous lines indented with same number of spaces forms a block.
* Semicolon ( ; ) allows multiple statements on the single line.
* A group of individual statements used to make a single code block are called suites.

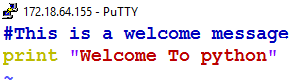
**1.2 Python Comments**

Python allows in-code documentation by using comments.

We can comment portion of code in two ways.

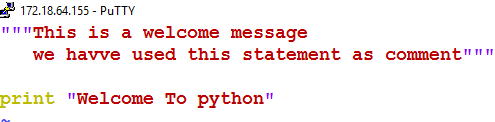
1. **Starting a line with a #**  : If a #is used in the beginning of a line, Python will consider the rest of the line as a comment.

**Example :**

****

**2. Using docstrings** : Python docstrings provide extended documentation capabilities. It can be a single line or multiple lines comment. Line or lines to be commented are started and ended with triple quotes.

**Example:**

****

**1.3 Identifiers in Python**

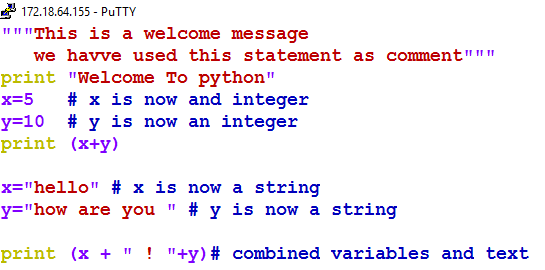
* Identifier is a name used to reserved memory locations to store values of a variable, function, class, module or other object.
* An identifier starts with a letter A to Z or a to z or an underscore (\_) followed by zero or more letters, underscores and digits (0 to 9).
* Python does not allow punctuation characters such as @, $, and % within identifiers.
* Python is case sensitive; hence, Sum and sum are two different identifiers.
* In a class, an identifier with a single leading underscore indicates that the identifier is private and an identifier with two leading underscores indicates a strongly private identifier.

**1.4 Python Variables**

An identifier whose value changes during program execution.

* Variables do not need to be declared with any particular type.
* Same variable can be used to set values of different data types.
* A variable is created the moment a value is assigned to it.
* Print statement is used to output variables.
* One variable can be added to another variable using + character.

**Example:** A sample script test.py is provided below.



**1.5 Python Data Types**

Following are the standard data types in python

* Numbers
* String
* List
* Tuple
* Dictionary

**1.5.1 Numbers**

* This data type stores numeric values.
* We can create a Number objects by assigning a value to a variable.

**Syntax:**

Var1 = 10

Count = 0

* The reference to a number object can be deleted by using the del statement.

**Syntax:**

del Var1,Count

* There are three number types.

**int :**  It is a whole number, positive or negative, without decimals, of unlimited length.

**Example :**

X = 20

Y = 12344556677

Z = -201

**float :** It is a number, positive or negative, containing one or more decimals.

**Example :**

X = 20.567

Y = - 12. 5

Z = -12.5e100

**complex :** numbers are written with a "j" as the imaginary part.

**Example :**

X = 20.567j

**Note :** String, List, Tuples and Dictionaries are discussed in the later section.

**1.5 Python Operators**

Operators are the constructs used to perform operations on variables and values.

**1.5.1 Operator Types**

* Arithmetic Operators
* Comparison or Relational Operators
* Assignment Operators
* Logical Operators
* ​​​​​​​​​​​​​​Bitwise Operators
* ​​​​​​​​​​​​​​​​​​​​​Membership Operators
* Identity Operators

**1.5.1.1 Arithmetic Operators**

Arithmetic Operators are used with numeric values to perform common mathematical operations.

Operators are :

       +  Addition

       -  Subtraction

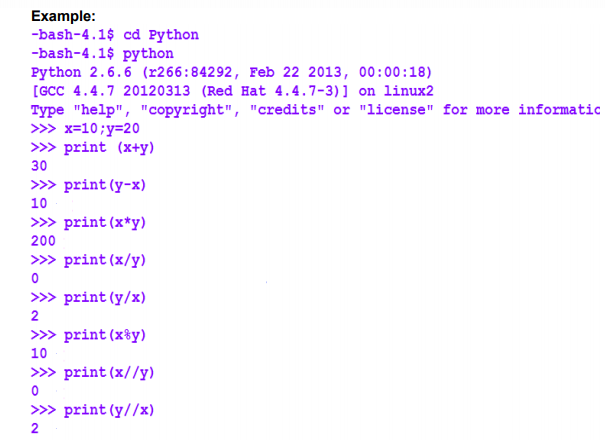
        /  Division

       \*  Multiplication

       %  Modulus

      \*\*  Exponentiation

        //  Floor Division



**1.5.1.2 Comparison or Relational Operators**

Comparison operators are used to compare two values.

Operators are:

== Equal

!= Not Equal

> Greater than

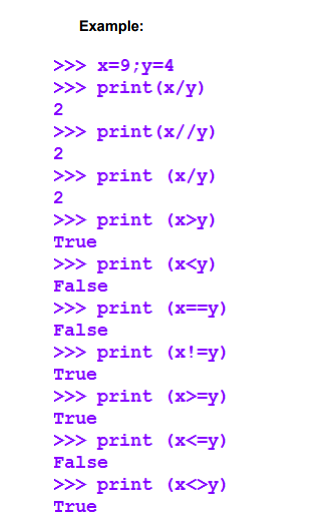
< Less Than

<> Not Equal

!= Not Equal

>= Greater Than Equal

<= Less Than Equal



**1.5.1.3 Assignment Operator**

Assignment operators are used to assign values to variables.

Operators are:

= assigns a value to a variable

+= adds right operand to the left operand and assign the result to left operand

-= subtracts right operand from the left operand and assign the result to left operand

\*= multiply right operand from the left operand and assign the result to left operand

/= divides left operand with the right operand and assign the result to left operand

%= returns the remainder when left operand is divided by the right operand and assigns the result to the left operand.

//= divides left operand with the right operand and assign the floor value result to left operand.

\*\*= calculate exponent value using operands and assign the result to the left operand.

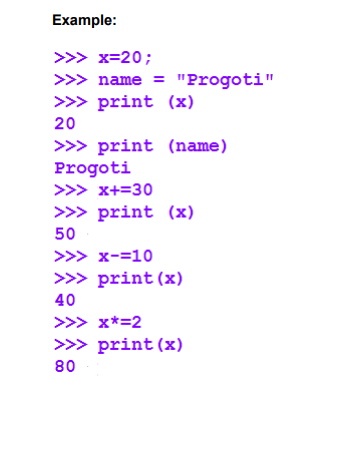
&= performs AND on operands and assign value to left operand

|= performs OR on operands and assign value to left operand

^= performs bitwise XOR on operands and assign value to left operand.

>>= performs bitwise right shift on operands and store values on left operand

<<= performs bitwise left shift on operands and store values on left operand



**1.5.1.4 Logical Operators**

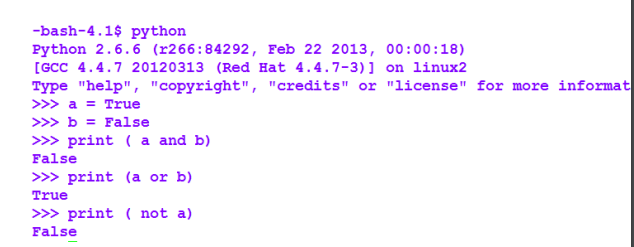
 These operators are used to combine conditional statements

Operators are:

and  - returns true if both the statements are true

or -  returns true if either of the statement is true not reverses the result

Example:



**.1.5.1.5 Bitwise Operators**

These are used to perform bit-by-bit operations.

Operators are:

&   bitwise AND, sets each bit to 1 if both bits are 1

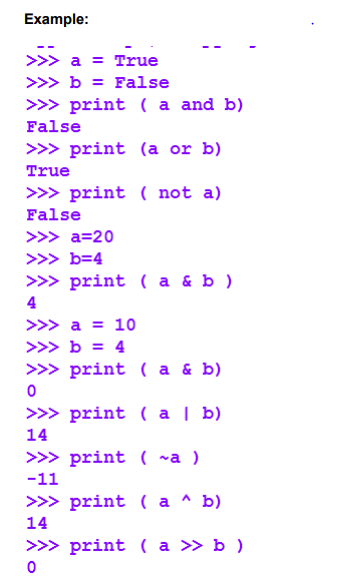
|    bitwise OR, sets each bit to 1 if one of the bits is 1

^   bitwise XOR, sets each bit to 1 if only one of the two bits is 1.

~   bitwise NOT, inverts all bit

<<  bitwise shift left

>> bitwise shift right



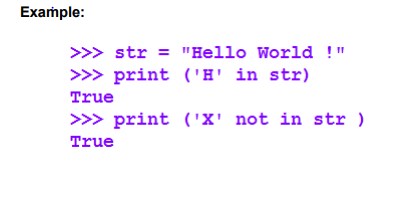
**1.5.1.6 Membership Operator**

These operators are used to test if a sequence is present in an object.

Operators are:

in               returns True if it finds a value in a specified sequence & False otherwise.

not in         returns True if it doesn't find a value in a specified sequence & False otherwise.

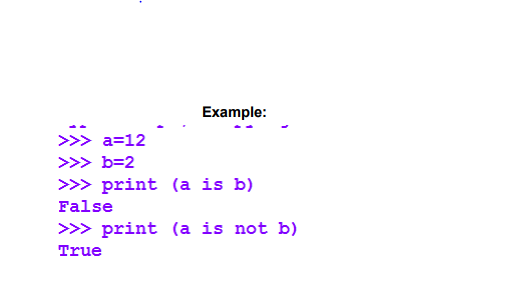


**1.5.1.7 Identity Operator**

 These operators are used to compare memory locations of two objects.

Operators are:

is returns true if both the operands are identical , that is, two values are located in the same part of memory location is not returns true if operands are not identical.



**1.5.2 Operator precedence & Associativity**

To evaluate an expression comprising of several operators, python has well-defined rules.

For example, multiplication and division have a higher precedence than addition and subtraction.

Using parentheses, we can explicitly override the precedence rules.

 For the detailed table stating the precedence of operators, use the following link. <https://www.programiz.com/python-programming/precedence-associativity>

In case of some operators, the precedence is same. When we use two operators with the same precedence in an expression, associativity helps in determining the order of their execution. Almost all the operators have left-to-right associativity.

# ****1. Conditional Constructs****

Conditional constructs are used to perform different computations or actions depending on whether the condition evaluates to true or false. The conditions usually uses comparisons and arithmetic expressions with variables. These expressions are evaluated to the Boolean values True or False. The statements for the decision taking are called conditional statements, alternatively known as conditional expressions or constructs.

## ****1.1 Types of Conditional Statements****

To understand the use of different conditional constructs in Python.

* If Statement
* If .. Else Statement
* If .. Elseif .. else statement
* Nested if statement
* Switch .. case statement

#### 

## ****1.1.1 If statement****

The if statements in Python. It is made up of three main components:

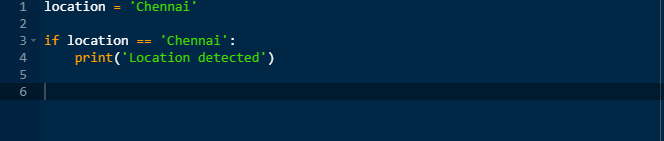
* the **if** KEYWORD itself,
* an EXPRESSION that is tested for its true value,
* a CODE SUITE to execute if the expression evaluates to non zero or true.

Syntax

**if expression:**

**expr\_true\_suite/statements**

The suite of **if** clause, (*expr\_true\_suite/statements)* will be executed only if the above conditional expression results in a boolean TRUE value. If the conditional expression results in a boolean FALSE value no output is obtained or the execution resumes at the next statement following the suite.



Few important things to note about **if** statements:

* The colon (:) is significant. It separates the **header** of the **compound statement** from the **body**.
* The line after the colon must be indented. It is standard in Python to use four spaces for indenting.
* All lines indented the same amount after the colon will be executed whenever the BOOLEAN\_EXPRESSION is true.

## ****1.1.2 if .. else statement****

Like other languages, Python features an else statement that can be paired with an if statement. The else statement identifies a block of code to be executed if the conditional expression of the if statement resolves to a false Boolean value.

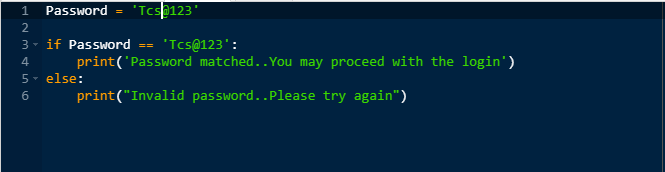
***Syntax :***

**if expression:**

**statement(s) //if condition evaluates to be true**

**else:**

**statement(s) //if condition evaluates to be false**



## ****1.1.3 If .. elif .. else statement (Chained conditions)****

elif is the Python else-if statement. It allows one to check multiple expressions for truth value and execute a block of code as soon as one of the conditions evaluates to be true. Like the else statement, the elif statement is optional. Unlike else, there can be an arbitrary number of elif statements following an if.

***Syntax :***

**if expression1:**

**statement(s) //if expression1 evaluates to true**

**elif expression2:**

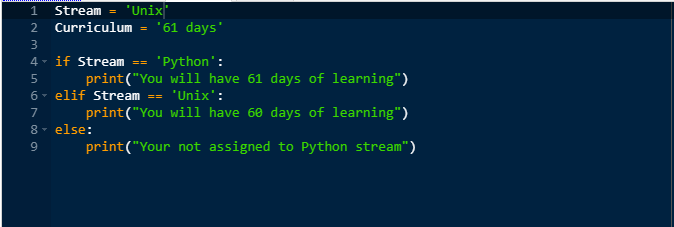
**statement(s) //if expression2 evaluates to true**

**elif expression3:**

**statement(s) //if expression3 evaluates to true**

**else:**

**statement(s) //if none of the above conditions are true**



## ****1.1.4 Nested If statements****

In Python one if condition can also be nested within another if condition. Indentation is the way to figure out the level of nesting.

***Syntax :***

**if condition:**

**statements**

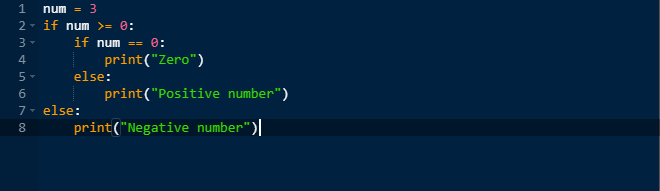
**else:**

**if condition:**

**statements**

**else:**

**statements**



## ****1.1.5 Switch case statements****

A switch statement is a multiway branching statement that compares the value of a variable to the values specified in case statements. It is a control statement to minimize a bunch of If - elif statements in a program. But, Python does not have a switch or case statement. To get around this fact, Python uses dictionary mapping to implement switch statement in Python.

## *https://g91.tcsion.com/per/g91/pub/2030/LX/ckeditor_assets/pictures/2030/2427/image_8632_original.png*

This is a Function to convert number into string. Switcher is dictionary data type here

get() method of dictionary data type returns value of passed argument if it is present in dictionary otherwise second argument will be assigned as default value of passed argument.

**2. Iterative Constructs**

A iterative construct is a structure which allows a block of statements to be executed repeatedly.

**2.1 Types of Iterative Statements**

The iterative constructs provided by python are:

* while iterative construct
* for iterative construct

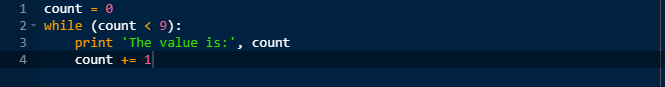
**2.1.1 while Iterative Construct**

The while statement in Python is a conditional looping statement. It is so called the entry controlled looping. The suite in a while clause will be executed continuously in a loop until the given condition is no longer satisfied.  
Since the test for exit of the loop is performed before the first instruction in the loop, based on the condition it may not execute for a single time. The variables appearing in the logical expression are initialized prior to the beginning of the while loop. This is  
mandatory as the logical expression in the while loop is tested before the loop is executed.

**Syntax**

while expression :  
suite\_to\_repeat

The suite\_to\_repeat clause of the while loop will be executed continuously in a loop until expression evaluates to Boolean False.

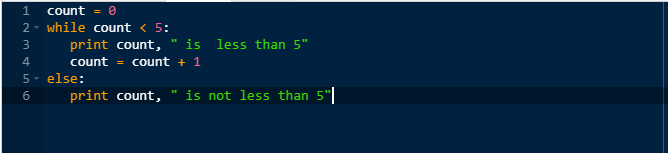


Output:

The value is: 0  
The value is: 1  
The value is: 2  
The value is: 3  
The value is: 4  
The value is: 5  
The value is: 6  
The value is: 7  
The value is: 8

**2.1.1.1 while loop with an else**

Python supports an else statement with a loop statement. When an else statement is used with a while statement, the else statement is executed when the expression evaluates to Boolean False.



Output:  
0 is less than 5  
1 is less than 5  
2 is less than 5  
3 is less than 5  
4 is less than 5  
5 is not less than 5

***Note : There is no do..while loop in Python.***

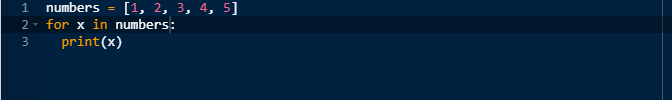
**2.1.2 for iterative Construct**

for statement represents the single most powerful looping construct in Python. The for loop in Python is more like foreach iterative type loop in a shell scripting language. Python's for takes an iterable and traverses each element once and terminates when all the items are exhausted.

**Syntax:**

for iter\_var in iterable:  
    suite\_to\_repeat

With each loop, the iter\_var iteration variable is set to the current element of the iterable (sequence, iterator, or object that supports iteration), presumably for use in suite\_to\_repeat.



Output:  
1

2

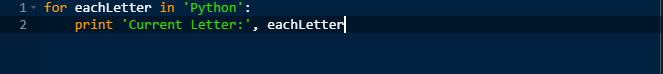
3

4

5

**2.1.2.1 For loop with Sequence Types:**

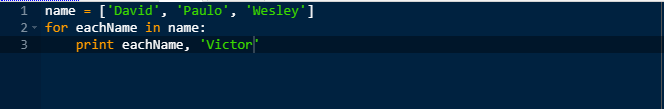
For loop works well with differenty sequence types in Python. The below example will include string, list and tuple types.



Output:  
Current Letter: P  
Current Letter: y  
Current Letter: t  
Current Letter: h  
Current Letter: o  
Current Letter: n

When iterating over a string, the iteration variable will always consists of only single characters(strings of length 1).

**2.1.2.2 Iterating with Sequence Item**



Output:  
David Victor  
Paulo Victor  
Wesley Victor

In the above example, a list is iterated over, and for each iteration, the eachNamevariablecontains the list that we are on for that partricular iteration of the loop.

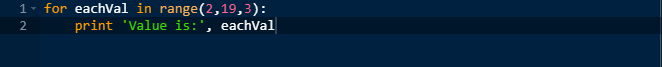
**2.1.2.3 range() function with for statement**

The built in function range() can turn the foreach-like for loop to the traditional type of for loop.

**Syntax:**

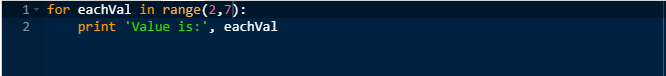
range (start\_value, end\_value, increment\_value)

range() will return a list where for any k, start\_value <= k < end\_value and k iterates from  start\_value to end value in increment of increment\_value. Increment value cannot be 0, or an error will occur.



Output:  
Value is: 2  
Value is: 5  
Value is: 8  
Value is: 11  
Value is: 14  
Value is: 17

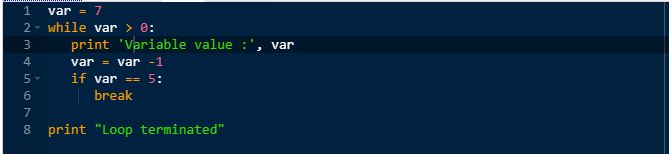
If the incremental\_value is ommited and only two arguments are given, increment\_value takes a default value of 1.



Output:  
Value is: 2  
Value is: 3  
Value is: 4  
Value is: 5  
Value is: 6

**2.1.3 Break statements**

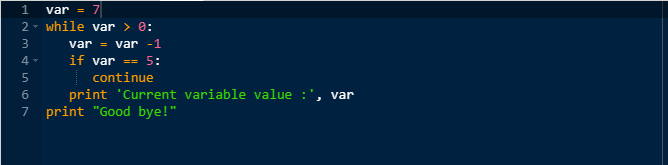
With break statement in Python,the current loop is terminated and the execution is resumed at the following statement. The most common use for break is when some external condition is triggered, requiring a hasty exit from a llop. This could with both while  
and for loop.



Output:  
Variable value : 7  
Variable value : 6  
Loop terminated

**2.1.4 Continue statement**

Whenever a continue statement in Python is encountered it re-starts a loop, skipping the following statements in the block. It could be used with both while and for loops.The while loop is conditional and the for loop is iterative, so using continue is subject to same requirements before the next iteration of the loop can begin. Otherwise the loop will  
terminate normally.



Output:  
Current variable value : 6  
Current variable value : 4  
Current variable value : 3  
Current variable value : 2  
Current variable value : 1  
Current variable value : 0  
Good bye!

**2.1.5 Pass statement**

One python statement not found in C is the pass statement. The pass statement does absolutely nothing. Its a true NOP, i.e No Operation assembly.It is useful in places where the code will eventually go, but has not been written yet.



This is more helpful during the development or debugging stages because youwant the structure to be there while the code is being created, but you do not want it to interface with other parts of the code. In places where you want nothing to be executed,  
pass is a good tool to be used.

# 1. Functions

Functions are construct to structure programs and are useful to utilize code in more than section in a program. It increases reusability of code and removes redundancy.

#### **1.1 Basic syntax**

A function is defined using the def keyword in python. The basic syntax is

**Syntax:**

def function\_name(parameters):

function body (statements)

The function body consist of indented statements. To end the function body, the indent is to be ended. Every time, a function is called the function body is executed. The parameters in the function definition are optional.

A function may have a return statement which returns a result. Once the return statement is executed in the function body the function is ended.

**Example:**

**>>> def square(x):**

**... return x\*\*2**

**...**

#### **1.2 Calling a function**

The defined functions are needed to be called to execute the function body.

A function is called by its name by passing the required values as defined as parameters in the function definition.

**Syntax:**

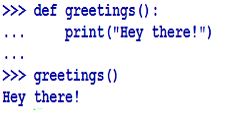
>>> function\_name(value)

If we want to find the square of 9, then we can call the function like:

>>> square(9)

81

**Examples:**

****

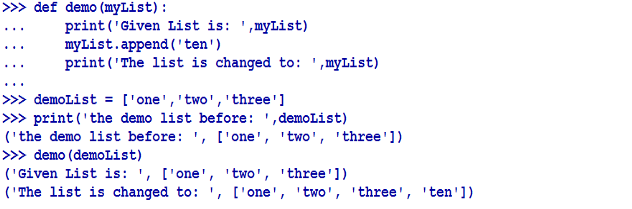
****

**1.2.2 Passing Parameters in a function by reference**

Here, a reference of actual parameter is passed to the function.

The advantage of it is that whenever the parameter value is modified the change reflects back in the function call.

**Example:**

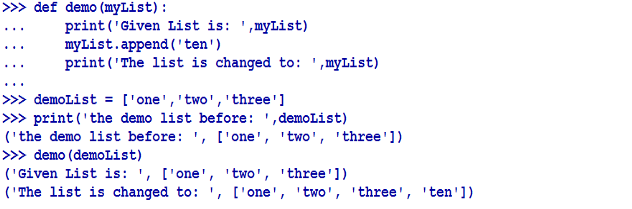
****

***Note:****In Python, scalar values are passed by-value. Lists and other objects are passed by reference.*

**1.3 Arguments**

The values passed in the parameters of a function are called arguments. The arguments provided in the function definition should match while calling the function. Or else, it will result in error.

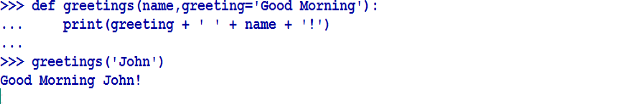
Example:



**Default Arguments:**

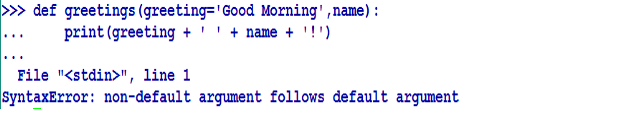
Function arguments can be provided with default values in function definition.

**Example:**

****

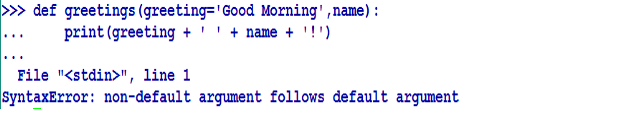
Here name does not have a default value. So it is going to take the value that is provided during the function call. But greeting has a default value and hence not needed to be provided with any value while calling the function.

**Note:**If a function has default arguments, all the arguments to its right are to provided as default arguments i.e. no non-default arguments can be present once a default argument is provided.

****

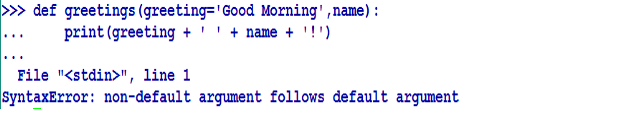
**Python Keyword Arguments:**

When a function is called using some values, the values are assigned to the arguments according to their position.



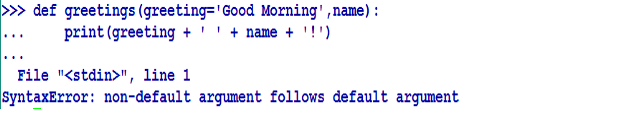
Here , in the above function, 'Hi' is assigned to greeting and 'Mark' is assigned to name when the function greetings is called.

Python allows functions to be called using keyword arguments. When we call functions using keyword arguments, the order of the parameters can be changed.



It is also possible to have positional arguments with keyword arguments while calling the function.

However, the positional arguments cannot be placed after the keyword argument.

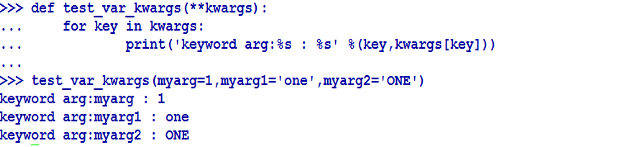


#### Variable length Arguments:

In Python, it is possible to provide variable arguments in a function as parameters. The single asterisk form (\*args) is used to pass a non-key worded, variable-length argument list, and the double asterisk form is used to pass a key worded, variable-length argument list.

**Example:**

****

****

**2. Scripts**

A script is a simple program, stored in a plain text file.

In Python, we can write programs through an interactive session in Python interpreter. However, it has the disadvantage that, if we quit from the interpreter and launch it again, the code we have made are lost.

We can create a Python script to solve this problem.

* A script is a Python code file created using any text editor.
* By convention, files are stored with an extension .py.
* Python interpreter runs python scripts.

**2.1 Running Python Scripts:**

Python interpreter follows a multi-step process to run a script known as Python Execution Model.

1. Interpreter processes the statements of the script in a sequential fashion.
2. Compiles the source code to an intermediate format known as bytecode. Bytecode is a translation of the code into low level language which is platform independent. The main purpose of a bytecode is to optimize code execution. Once , a byte code is created, next time when the interpreter runs the code, it can bypass this step. This is applicable only for modules, not for executable scripts.
3. The runtime engine of Python, PVM(python virtual machine) iterates over the instructions of the bytecode to run them one by one.

**Syntax:**

**python**

**Example:**

**python testCode.py**

where , testCode.py is the file or script name having python code.

**2.2 Redirecting the Output Of Script:**

To store the output of the script in a file , we can redirect it to a file.

**Syntax:**

**python testCode.py > output.txt**

This process is called stream redirection.

**3. Modules in Python**

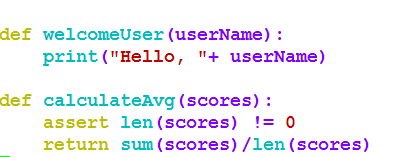
A module is a set of code written in a file having extension .py. Module contains function definitions and other statements. Module allows organize python code logically. We can make a group of related codes by creating a module. This makes it easier to understand the code and maintain the code. A module can define variables, functions and classes and runnable code.

**3.1 Creating a Module:**

To create a module, create a file with extension .py in an editor and write the code and save the file.

**Example:**

Created a file named myModule.py and defined two functions as below:



Two functions welcomeUser() and calculateAvg() are defined in the module.

**3.2 Using a Module:**

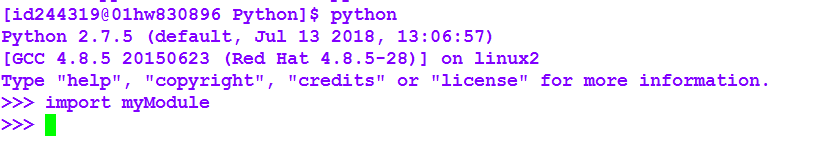
In the python interpreter, a module is imported using import statement.

**Syntax:**

import

**Example:**

To import the module myModule created above.



After importing the module, using the module name, we can access the functions or variables defined in that module.

**Example:**

**https://g91.tcsion.com/per/g91/pub/2030/LX/ckeditor_assets/pictures/2030/2675/image_26385_original.png**

Here, the function welcomeUser() defined in the module myModule is called passing “Progoti” as the argument to print the welcome message

We can use any Python source file as a module by executing an import statement in another Python source file. When the interpreter gets an import statement, it imports the module if it is present in the search path. A module is loaded only once regardless of number of times it is imported.

**3.3 Using Specific Attribute from Modules:**

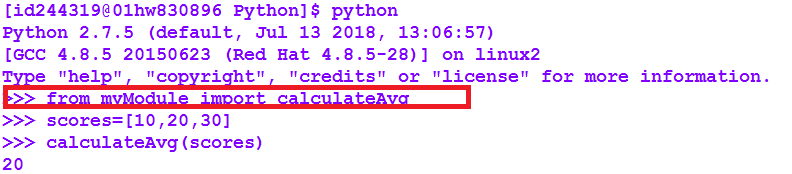
We can import only specific attributes from a module instead of the entire module. To do this we have to use the using from statement.

**Syntax :**

from import attribute1[,attribute2[,…nameN]]

**Example:**

To import the function calculateAvg(scores) from the module myModule to find the average of the scores stored in the list scores,



**Note:** When importing an attribute of a module using the from keyword, do not use the module name when referring to elements in the module. For E**xample , in the above code , we have used calculateAvg(scores), not mymodule.calculateAvg(scores)**

**Using the dir() Function:**

Built in function dir() lists the names of the attributes defined in a module.

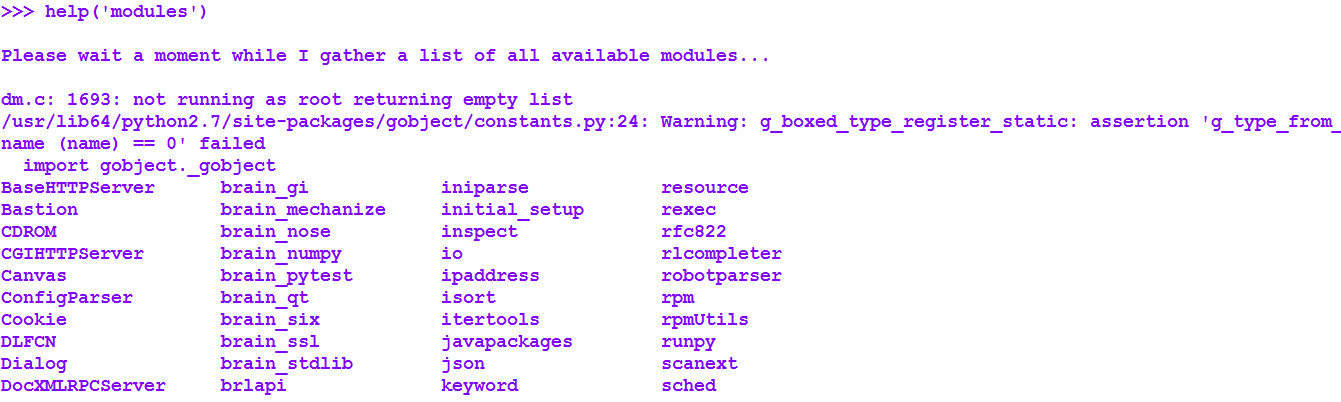
**Example:**

****

**3.4 Built-in Modules:**

Python has a collection of built-in modules. Programmers can use these modules as and when required. Using help command, we can get the list of available built-in modules.

**Example:**

****

Each built-in module contains resources for certain system-specific functionalities such as OS management, disk IO, etc. The standard library also contains many Python scripts (with the .py extension) containing useful utilities.

**4. Packages**

* A package is a bundle of Python modules.
* A package is a directory with Python files.
* A Python package can have sub-packages and modules.
* A package must have a file with the name \_\_init\_\_.py.
* It is possible to put several modules into a Package.
* Packages help in structuring files in a hierarchical structure that defines a single Python application environment

**4.1 Creating a Package:**

Following are the steps to be followed to create a package.

1. Create a directory. Name of the directory will be the name of the package.
2. Create a file named \_\_init\_\_.py inside the directory. Presence of this file in the directory let Python know that the directory is a package.
3. Create all Python files in this directory, which will be modules or sub modules of the package. This file is invoked when the package or a module in the package is imported. This can be used for execution of package initialization code, such as initialization of package-level data.

**4.2 Accessing the Packages:**

To access modules or functions from a package we have to use import statement.

Syntax:

We can use any of the following syntax.

* from import [,…]
* from import as
* import ]

**Example:**

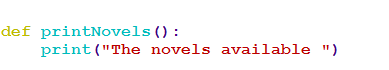
Let us create a package named Books and build three modules in it namely, Novel, Horror and General\_Knowledge.

1. First created a directory Book and go to the directory.

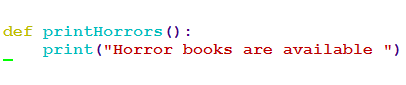
https://g91.tcsion.com/per/g91/pub/2030/LX/ckeditor_assets/pictures/2030/2679/image_62481_original.png

2. Now create the modules inside this directory by creating three files novel.py, horror.py and GK.py and define code in these files.

In the file novel.py, defined function printNovels() to print a message.



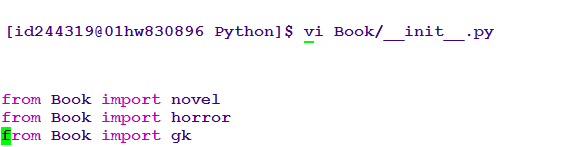
In the file horror.py, defined function printHorrors() to print a message.



In the file gk.py, defined a function printGKs() to print a message.

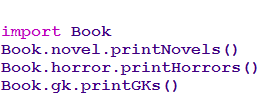
https://g91.tcsion.com/per/g91/pub/2030/LX/ckeditor_assets/pictures/2030/2683/image_52022_original.png

3. Finally, we create the file \_\_init\_\_.py  inside Book directory . To make all of your functions available when you will imported Book, you need to put explicit import statements in \_\_init\_\_.py as follows :

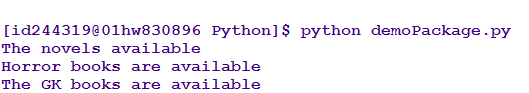


To use the package Book created, create a file demoPackage.py in the directory where the directory Book is present.

Now, to use the Book package,created the file demoPackage.py as below.

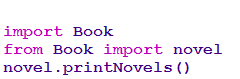


When we execute this file, the functions defined in the files included in the package are executed.



In case, the import statements are not included in the \_\_init\_\_.py file , we have to add them before we invoke the objects defined in the files included in the package.

For example, if we keep the \_\_init\_\_.py file empty for the Book package created and have to access the printNovels() function from the Book package , we have to define the script as below:



Note: In the above example, we have taken example of a single function in each file, but we can have multiple functions in each files, also define different Python classes in those files and then can create packages of those classes.

## 1. Sequences

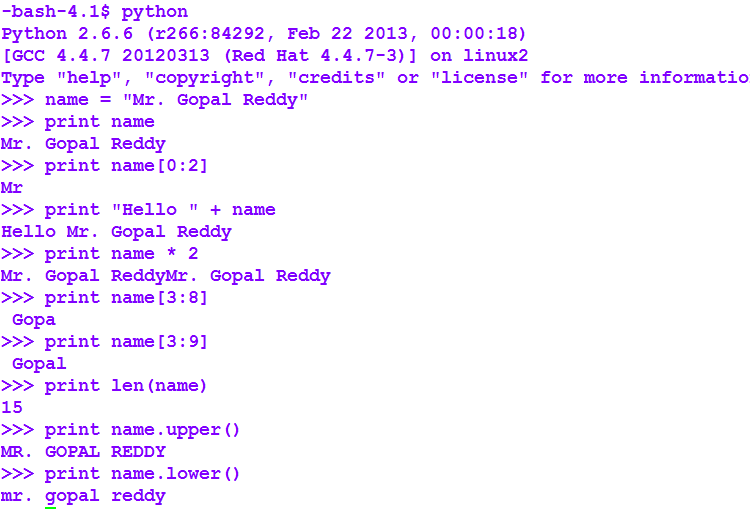
Sequences are the container data-types in Python which stores objects in a specific order. The objects can be accessed using their position or index. Indexes are numbered from 0 in Python.

There are six sequence types in Python  and we will discuss them one by one below.

## ****1.1.Strings****

* A contiguous set of characters represented in the quotation marks is a string.
* We can use either single quotation or double quotation marks.
* Strings in Python are arrays of bytes representing unicode characters.
* Subsets of strings can be taken using the slice operator ([ ] and [:] )
* + is the string concatenation operator.
* \* is the repetition operator.
* len() method returns the length of string
* upper() method converts the string to upper case
* lower() method converts the string to lower case

**Example:**

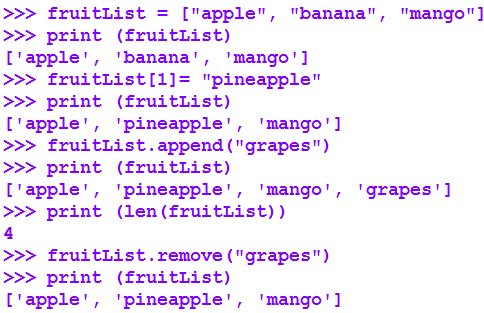


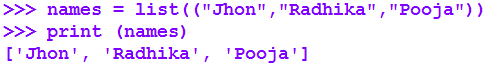
**1.2. Lists**

List is a compound data type.

* It contains items separated by commas and enclosed within square brackets ([]).
* Items belonging to a list can be of different data type.
* List is ordered and changeable.
* List allow duplicate data.
* List can be created by using the list constructor list().
* append() object method is used to add an item to the list.
* remove() object method is used to remove a specific item from the list.
* len() method is used to get a count of elements in the list.

**Example:**

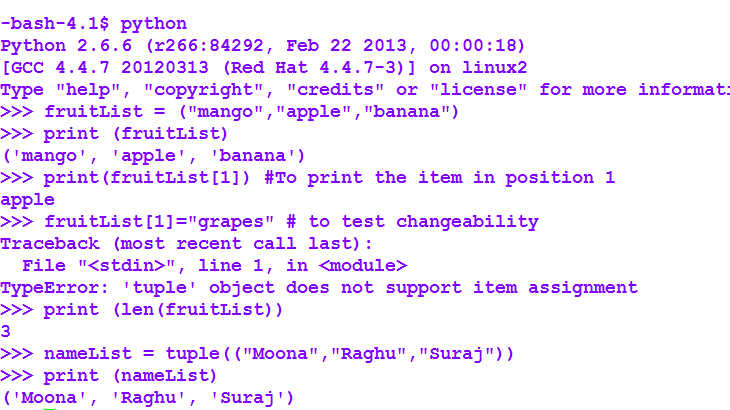
****

****

**1.3. Tuples**

* A tuple is an ordered and unchangeable list or collection.
* In Python tuples are written with round brackets.
* Items in the tuple are separated by commans.
* Can use tuple() constructor to make a tuple.
* len() method returns the number of items in a tuple.

**Example:**

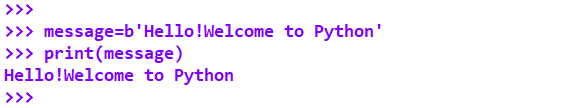
****

**1.4. Bytes**

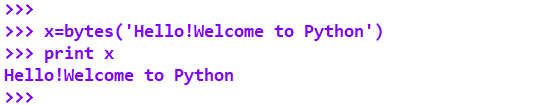
The bytes object is a sequence of small integers. The elements of a byte object are in the range 0 through 255, corresponding to ASCII characters. Like tuples, bytes is an immutable sequence. Bytes can be directly stored in the memory without encoding which is not in the case of strings. In strings, first we have to encode the data and then store it. A byte object can be created using the byte() function or using the prefix b.

**Example :**

1. Creating a byte object using the prefix b

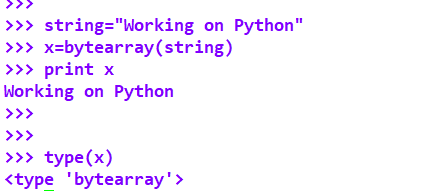


2. Creating a byte object using the byte() function



**1.5 Bytearray**

The main difference between byte array and byte sequence type is the  byte array is mutable anmd bytes are immutable. To crate a bytearray we can use the bytearray() function.



**1.6 Range**

The range()  type is an immutable sequence of numbers in Python within a given range.

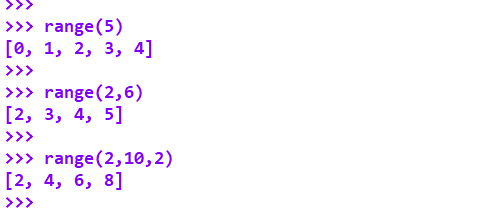
Range takes three aruments :

a. start : the starting integer of the sequence.Thsi si optional. if  no start is given, Python considers the start as 0.

b. stop : the integer before which the sequence of integers are to be considered.

c. step : step is optional parameter. It determines the value by which each integer in the sequence is to be inserted.

**Example :**

****

**1. Collections**

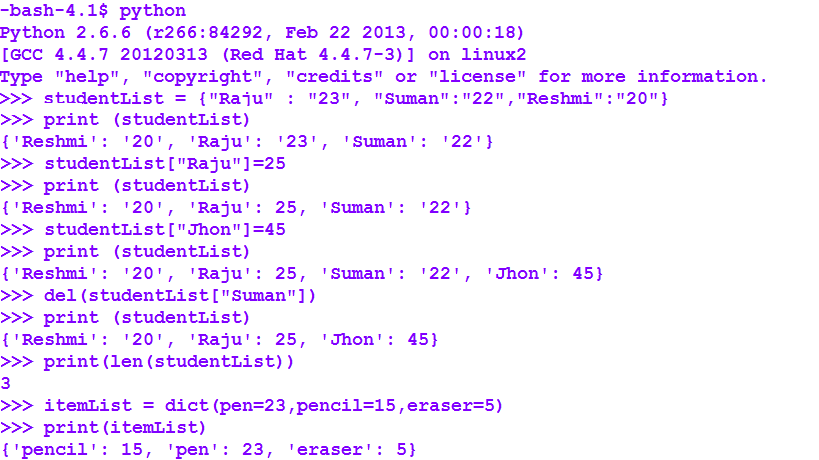
Collections are the container data-types which stores objects in an unordered way. Since there is no order maintained, no indexing for the elements are done.

Python offers two default collections and five high performing collections from the collection module. We are going to discuss them one by one.

**1.1 Dictionaries**

* A dictionary is an unordered collection.
* It is changeable and indexed using the key.
* Dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessed using square braces ([])
* len() method to returns the number of items.
* the dict() constructor can be used to make a dictionary
* We can add an item to the dictionary by using a new index key and assigning a value to it.
* Elements are stored in a dictionary in a key-value pair and the pair is unique.
* We can remove item from a dictionary using the del() function.

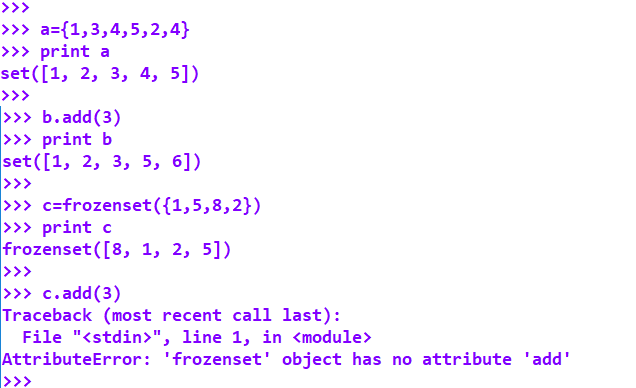
**Example:**

****

**1.2. Sets**

* A set is an unordered collection​
* It is iterable, mutable and has no duplicate elements.
* Sets are enclosed by curly braces ({ }).
* A  set can be created using the set constructor.
* Elements can be added to  a set using the add() method.
* A frozen set is an immutable object which can be created using the frozenset constructor.

**Example :**



**1.3 High Performance collection datatypes**

We have already discussed how we can use Python built in collections, dictionaries and sets to group multiple values together in a collection. Here, we are going to discuss about the collections module.

Collections module in Python, implements specialized container datatypes providing alternatives to Python’s general purpose built-in containers  list, set, dictionary and tuple.

To use the classes available in the collections module, we have to import the module as below.

**>>>import collections**

The classes available in this module are:

* Counter
* Deque
* Defaultdict
* Namedtuple
* Ordereddict

​​​​​​​

**1.3.1 Counter**

* Counter is a **container** included in the collections module of Python.
* It is a subclass of dictionary and is an unordered collection where elements and their respective count are stored as dictionary.
* It has a method called **elements()** that returns an iterator over elements repeating each as many times as its count.
* It has a method called**most\_common()** that returns most common elements and their counts from the most common to the least.

**Counter Initialization**

The constructor of counter can be called in any one of the following ways :

* With sequence of items

**Example**:

https://g91.tcsion.com/per/g91/pub/2030/LX/ckeditor_assets/pictures/2030/2447/image_20859_original.png

* With dictionary containing keys and counts

**Example:**

**https://g91.tcsion.com/per/g91/pub/2030/LX/ckeditor_assets/pictures/2030/2448/image_28448_original.png**

* With keyword arguments mapping string names to counts

**Example:**

**https://g91.tcsion.com/per/g91/pub/2030/LX/ckeditor_assets/pictures/2030/2449/image_15280_original.png**

In all three cases, the output is same.

Use of elements() and most\_common() methods

**elements()**method returns an iterator over elements repeating each as many times as its count. Elements are returned in arbitrary order. If an element’s count is less than one, [elements()](https://docs.python.org/3/library/collections.html#collections.Counter.elements) will ignore it.

**Example:**

**most\_common()** method return a list of the n most common elements and their counts from the most common to the least.

If n is omitted , or nor none , [it](https://docs.python.org/3/library/collections.html#collections.Counter.most_common)  returns all elements in the counter. Elements with equal counts are ordered arbitrarily.

**Example:**

**https://g91.tcsion.com/per/g91/pub/2030/LX/ckeditor_assets/pictures/2030/2450/image_1723_original.png**

In the above code snippet, created a counter object c and used the most-common() method to print the elements with their counts.

https://g91.tcsion.com/per/g91/pub/2030/LX/ckeditor_assets/pictures/2030/2450/image_1723_original.png

In the above code snippet, printed the first 2 most common elements.

**Mathematical Operations on Counter objects**

Following mathematical operations are provided for combining Counter  objects to produce multisets, i.e counters that have counts greater than zero. Each operation can accept inputs with signed counts, but the output will exclude results with counts of zero or less.

* + Addition and subtraction combine counters by adding or subtracting the counts of corresponding elements.
  + Intersection and union return the minimum and maximum of corresponding counts.

Created two counter objects c and b as below:

https://g91.tcsion.com/per/g91/pub/2030/LX/ckeditor_assets/pictures/2030/2451/image_31585_original.png

Counter c is:

https://g91.tcsion.com/per/g91/pub/2030/LX/ckeditor_assets/pictures/2030/2452/image_9526_original.png

Counter b is:

https://g91.tcsion.com/per/g91/pub/2030/LX/ckeditor_assets/pictures/2030/2453/image_46486_original.png

Let us implement addition and subtraction to combine these two counters as below.

https://g91.tcsion.com/per/g91/pub/2030/LX/ckeditor_assets/pictures/2030/2454/image_82876_original.png

After addition the count of a is 4 , count of b is 3 and now we have c as 1 and d as 1.

https://g91.tcsion.com/per/g91/pub/2030/LX/ckeditor_assets/pictures/2030/2457/image_71980_original.png

After subtraction, since the count of ‘a’ and ‘b’ has become less than one, are not in the resultant object.

**1.3.2 Deque**

Double Ended Queue or deque is used as the data structure stack. In situations where there is a requirement of structuring the data in a way that facilitates quickly appending to either end or retrieving from either end then a deque can be used.

Deque object supports following set of methods :

* **append()** - used to **insert**the value in its argument to the **right end** of deque.

* **appendleft()** - used to**insert** the value in its argument to the **left end** of deque.

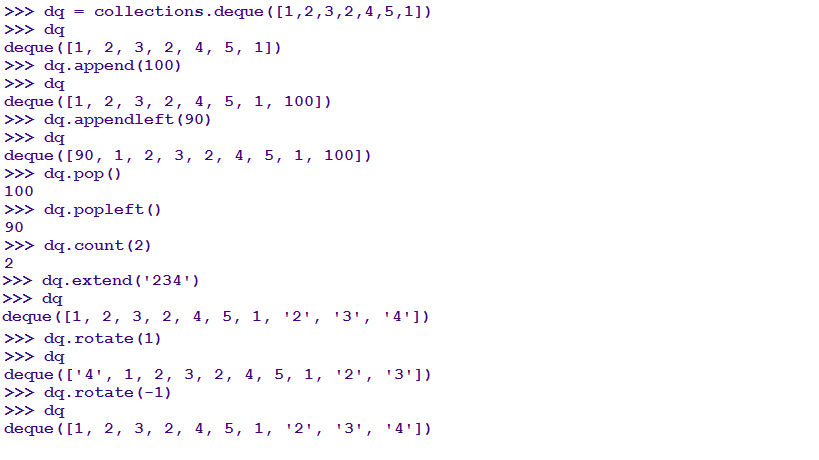
* **pop()** - used to**delete** an argument from the**right end** of deque.

* **popleft()** - used to**delete** an argument from the**left end** of deque.

* **clear()-** used to remove all elements from the deque leaving it with length 0.

* **copy**() – used to create a shallow copy of the deque.
* **count**(*x*)- used to count the number of deque elements equal to *x*.
* **extend**(iterable) – used to extend the right side of the deque by appending elements from the iterable argument.
* **extendleft**(iterable) – used to extend the left side of the deque by appending elements from iterable. The series of left appends results in reversing the order of elements in the iterable argument.
* **remove**(value) - used to remove the first occurrence of *value*. If not found, raises a [ValueError](https://docs.python.org/3/library/exceptions.html" \l "ValueError" \t "_blank).
* **insert(i, a)** :- used to  **insert the value** mentioned in arguments(a) **at index(i)** specified in arguments.
* **maxlen –**used to get the maximum size of a deque.

**Example:**

****

**1.3.3 Defaultdict**

defaultdict  is a container in the collections class of Python . This container is similar to the usual dictionary container having only one difference. Unlike dictionary, defaultdict allows to specify the default type of the value.

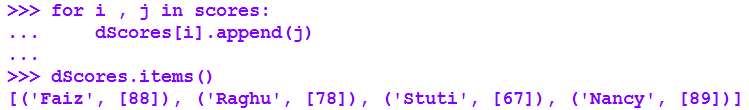
In defaultdict , we pass in the default type upon instantiation. Then we can immediately begin setting values even if the key is not yet set.

**Example:**

Creating a defaultdict using list as the default factory.

https://g91.tcsion.com/per/g91/pub/2030/LX/ckeditor_assets/pictures/2030/2462/image_50245_original.png

In this example , created **dScores** a defaultdict object using the default\_factory function which returns an empty list



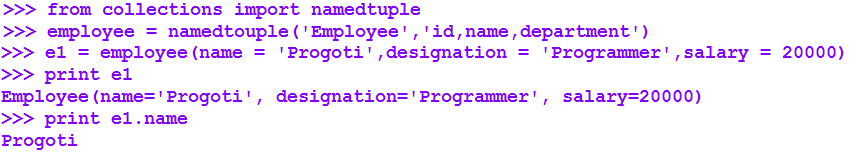
Next , in a for loop , using the  list.append() operation , attached the value to the new list. When keys are encountered again, the look-up proceeds normally (returning the list for that key) and the list.append() operation adds another value to the list.

**1.3.4 Namedtuple**

It is a Factory Function for Tuples with Named Fields. Named tuples help to have meaning of each position in a tuple. Using it, we can increase the readability of the code as the code becomes self-documenting.

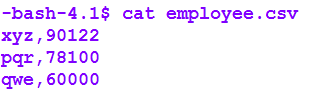
To instantiate, we pass in the name of the type we want to create. Then we pass in a list of field names.

**Example:**

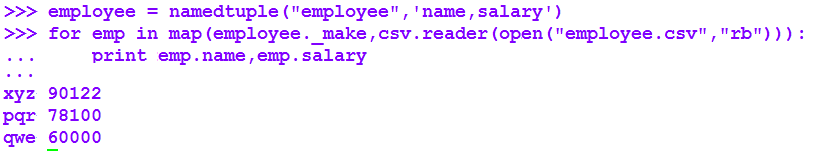
****

Using namedtuple, we can grab data from a csv .

In the below example , we have a .csv file “employee.csv” having values for employee name and salary.



Created a namedtuple and fetched data from the file as below:

\

**1.3.5 Ordereddict**

An OrderedDict is a dictionary that remembers the order of the keys that were inserted first. If a new entry overwrites an existing entry, the original insertion position is left unchanged. When we are iterating over the OrderedDict , the order will reflect the order in which the keys were added.

As it is a sub class of Dictionary, all the methods of dictionary are applicable for it.

**1. Regular Expressions**

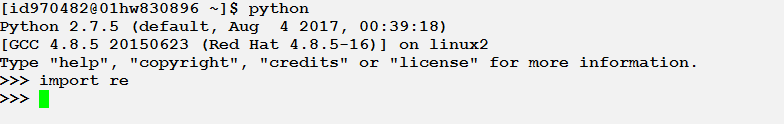
Regular expressions are used to check the existence of a pattern in a given sequence of characters. The pattern to be checked is the regular expression. Regular expressions are widely used in many applications to validate the user inputs like email-ids, passwords etc.

Regular Expressions in Python is very powerful. It is supported by the re module which needs to be first imported.

**1.1. Re module in Python**

The re library in Python has many functions which comes handy while using regular expressions.

But as said, the re module is to be imported first



Lets discuss some of them which will be very useful while using regular expressions.

**1.1.1 Match Function**

The match function checks the text(sequence) with the pattern and returns a matched object if they matches and a none when they do not match.

**Basic Syntax:**

re.match(pattern,sequence,flags=0)

**Example:**

****

**1.1.2 Search Function**

The search function scans for the pattern in the entire sequence and returns a matched object if the pattern in found in any position in the sequence. If the pattern is found nowhere in the sequence, then it returns None.

**Basic Syntax:**

re.search(pattern,sequence,flags=0)

**Example:**

https://g91.tcsion.com/per/g91/pub/2030/LX/ckeditor_assets/pictures/2030/2542/image_68572_original.png

**Note:** The difference between search and match is that match matches the entire pattern with the sequence but search matches the pattern if it is anywhere present in the sequence

https://g91.tcsion.com/per/g91/pub/2030/LX/ckeditor_assets/pictures/2030/2543/image_1213_original.png

**​​​​​​1.1.3. Substitute Function**

This function substitutes the found pattern with the given pattern and replaces it with the given new pattern and returns the new sequence. If the given pattern is not found then it is not replaced and returned as it is.

**Basic Syntax:**

re.sub(pattern, replace, string, count=0, flags=0)

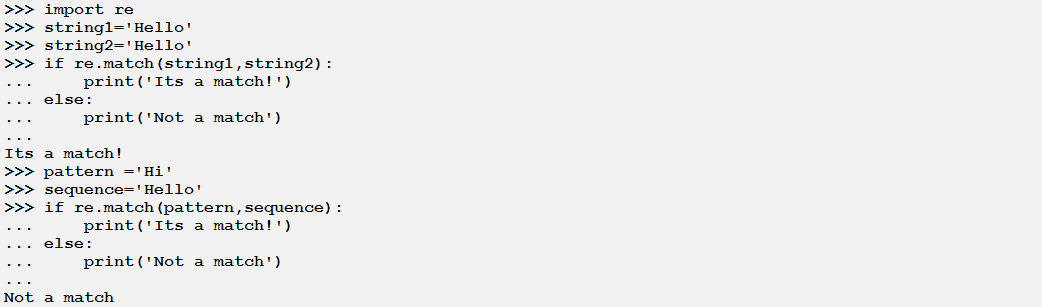
**Example:**

**https://g91.tcsion.com/per/g91/pub/2030/LX/ckeditor_assets/pictures/2030/2544/image_59473_original.png**

**1.2 Basic Patterns**

Basic Patterns are the easiest to use in Python. Here ordinary characters are used as regular expressions. The ordinary characters matches themselves and no other meaning is interpreted. They do not have any special meaning in their regular expression syntax.

Let us try to use regular expressions to match two patterns and see the output



**1.3 Wildcard Patterns(Special Characters)**

Special characters are those characters which do not match themselves but have some other meaning while matched with a sequence.

**1.3.1. List of Special Characters**

Below a list of most widely special characters are provided with some examples.

As it is already discussed, if a match object is returned on execution of the statement. It means it a match.

|  |  |  |
| --- | --- | --- |
| **Character** | **Name of the Character** | **Use** |
| . | A Period | Matches any single character in a sequence  https://g91.tcsion.com/per/g91/pub/2030/LX/ckeditor_assets/pictures/2030/2547/image_87522_original.png |
| w | Lowercase w with a backslash | Matches any single letter, number or underscore in a sequence.  https://g91.tcsion.com/per/g91/pub/2030/LX/ckeditor_assets/pictures/2030/2548/image_52554_original.png |
| \W | Uppercase w with a backslash | Matches any other character in a sequence other than the characters matched by \w(any other character than letter, number or underscore)  https://g91.tcsion.com/per/g91/pub/2030/LX/ckeditor_assets/pictures/2030/2549/image_75086_original.png |
| \s | Lowercase s with a backslash | Matches a single white space character in a sequence.  https://g91.tcsion.com/per/g91/pub/2030/LX/ckeditor_assets/pictures/2030/2550/image_92080_original.png |
| \S | Uppercase s with a backslash | Matches any other character that is not part of \s  https://g91.tcsion.com/per/g91/pub/2030/LX/ckeditor_assets/pictures/2030/2551/image_76552_original.png |
| \t | Lowercase t | Matches tab |
| \n | Lowercase n | Matches newline |
| \d | Lowercase d | Matches digits from 0-9 |
| ^ | Caret | Matches a pattern a the beginning of a sequence  https://g91.tcsion.com/per/g91/pub/2030/LX/ckeditor_assets/pictures/2030/2552/image_82520_original.png |
| $ | Dollar | Matches a pattern at the end of the sequence  https://g91.tcsion.com/per/g91/pub/2030/LX/ckeditor_assets/pictures/2030/2553/image_87604_original.png |
| [a-zA-Z0-9] |  | Matches any letter from a to z or A to Z or any number between 0 and 9 |

**1.3.2. Repetitions**

Using Python we can handle repetitions of characters using regular expressions very well.

Below are a list of characters which helps us to handle repetitions.

|  |  |
| --- | --- |
| **Character** | **Use** |
| + | Checks for the presence of one or more characters as the same character as its left  https://g91.tcsion.com/per/g91/pub/2030/LX/ckeditor_assets/pictures/2030/2554/image_51978_original.png |
| \* | Checks for the presence of zero or more characters as the same character as its left  https://g91.tcsion.com/per/g91/pub/2030/LX/ckeditor_assets/pictures/2030/2555/image_59538_original.png |
| ? | Checks for the presence of zero or more characters as the same character as its left.  https://g91.tcsion.com/per/g91/pub/2030/LX/ckeditor_assets/pictures/2030/2556/image_29814_original.png |
| {x} | Repeats at least x number of times |
| {x,} | Repeats at least x number of times or more. |
| {x,y} | Repeats at least x number of times but not more than y number of times. |

Let us take an example how to use the last three characters to handle repetition.

Suppose we want to validate the contact number provided. A contact number has to be of 10 digits so we can validate it as following.

https://g91.tcsion.com/per/g91/pub/2030/LX/ckeditor_assets/pictures/2030/2557/image_93316_original.png

**1.4 Greed and Non-Greedy Matching**

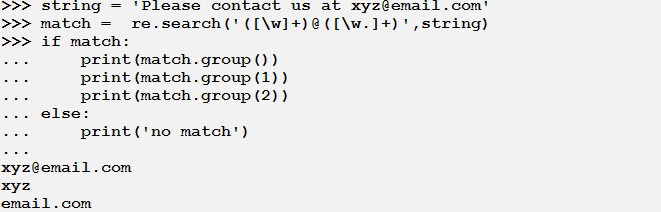
When a character matches as many characters as possible, then it is called a greedy match. If only one or zero characters are matched by a special character, then it is a non-greedy match.

\* and + characters are considered to be greedy.

**1.5 Group Feature**

The group feature is very useful when we want to select some portion of the matched text. Part of the regular expression which is bounded by parentheses are called groups

Suppose we want to view only the email address from a given sequence. Let us see how to do it using group.



**1.6. References Link**

<https://docs.python.org/3/howto/regex.html>

<https://www.tutorialspoint.com/python/python_reg_expressions.htm>

h[ttps://www.regular-expressions.info/python.html](https://www.regular-expressions.info/python.html" \t "_blank)

<https://regexone.com/references/python>

**1. Comprehensions**

Comprehensions are constructs that allow to build sequences from other sequences.

Python supports four types of comprehensions:

* List Comprehensions
* Dictionary Comprehensions
* Set Comprehensions

**1.1.List Comprehensions**

List comprehension allows creating lists.

**Syntax:**

= [ output\_expr for var in input\_list (Predicate expression) ]

Here,

output list -> the new list to be created

outpt\_expr -> Expression producing elements of the output list from members of the Input Sequence

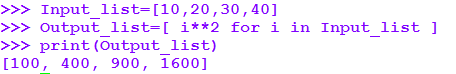
var -> a variable representing the members of the input sequence

input\_list -> the input list

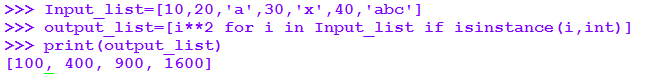
Predicate expression -> a condition to be satisfied for the output element created. This is optional.

**Example:**

We want to create an output list, which contains squares of all the numbers of an input list.



Our input list may have some non-numeric values also. In that case the expression used will not work. Hence we can put a condition in the comprehension to check whether the element in the list is a number or not and if so the expression will be executed and element for the output element will be generated.



This time we have used a predicate to check whether the element of the input list is an integer or not.

Working of the comprehension:

* Here, we have used an iterator. Iterator iterates through each member I of the input sequence, input\_list
* The predicate checks if the member is an integer
* If the member is an integer then it is passed to the output expression, squared to become a member of the output list.

List comprehensions can contain nested loops .

Syntax:

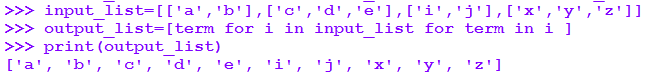
[ expression for i in list1 [if condition1]

for j in list2 [if condition 2]…

for n in listN [if condition N]]

Example:

Suppose we have to create a list by taking elements from a list of lists.



**1.2. Dictionary Comprehensions**

Dictionary comprehension is the method of creating a new dictionary from an existing dictionary.

We can say, using comprehension we can convert one dictionary to another dictionary.During this transformation, items within the original dictionary can be conditionally included in the new dictionary and each item can be transformed as needed.

The general syntax for dictionary comprehension in Python is as below:

**Output\_dict = {key:value for (key,value) in input\_dict.items()}**

Alternatively, we can use the following syntax if we want to have predicate.

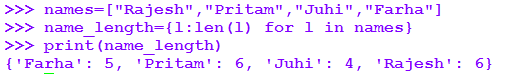
**Output\_dict = {key:value for (key,value) in input\_dict.items() if condition }**

Dictionary comprehension helps in writing compact code by substituting for loops and lamda

functions.

**Example 1:**

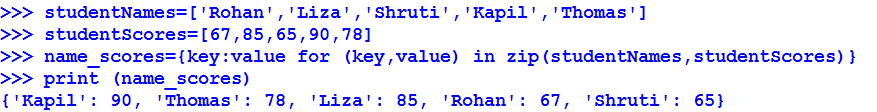
We have a list of names of person. We have to create a dictionary of name of the person as key and length of each name as value using dictionary comprehension.



Here, names in the input list ,name\_length is the dictionary created using comprehension.

**Example 2:**

Given two lists containing the names of students and their corresponding scores, construct a dictionary which maps the student names with their respective scores.



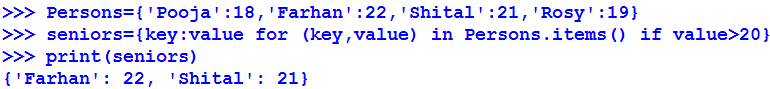
Here, studentNames list has list of names of students and studentScores list has scores of the students. Output dictionary ,name\_scores is created from studentNames and studentsScores lists using comprehension .

**Adding Conditionals to Dictionary Comprehension**

We can add conditions in dictionary comprehensions to create the dictionary by selecting items as per requirement from the input lists /dictionaries.

**Example:**

Suppose we need to create a new dictionary from a given dictionary of name:age of persons as key:value pairs but with names whose age is greater than 20



**Advantages of Dictionary Comprehension:**

Following are the advantages of using dictionary comprehwnsions:

* + Alternative to for loop -> Nested for loops makes code confusing and complex. Use of dictionary comprehensions are better in such situations and can simplify the readability and your understanding of the code.
  + Alternative to lamda functions -> Complex ,multiline lamda functions can be replaced with a single line of code using dictionary comprehensions.

**Nested dictionary comprehensions:**

A nested dictionary is a dictionary inside another dictionary that is a collection of dictionaries

into a single dictionary.

**Syntax:**

Out\_data = {outer\_k: {inner\_k: myfunc(inner\_v) for inner\_k, inner\_v in uter\_v.items()} for outer\_k, outer\_v in outer\_dict.items()}

**1.3. Set Comprehension**

A set comprehension produces a new set by generating a set of keys derived from elements in a source container for elements that obey a condition.

**Syntax:**

Output\_set= {expression(variable) for variable in input\_set [predicate][, …]}

Where,

**Output\_set :**the resultant new setto be created.

**expression :**An output expression producing members of the new set from members of the input set that satisfy the predicate expression.

**variable**: a variable representing members of an input set.

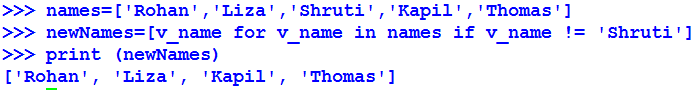
**input\_set**: represents the input data set.

**predicate:** expression acting as a filter on members of the input set

**[,…]]**:nested comprehension

Except the variable and input\_set, rest of the components are optional.

**Example:**



In the above example, a new set of names created from the existing set of names excluding the name Shruti.

## ****1. File Handling****

File handling in python plays an essential role to work with files either to write to a file or to read data from it. With file handling, operating on the data present in the file becomes very easy.

**1.1. File Open**

The built in function to open and create a file object is:

**Syntax:**

**file\_object = open(name [,mode [,bufsize]])**

**name**: It is the name of the file.

**mode**: mode in which the file has to be open.

**bufsize**: If the value is set to zero (0), no buffering will occur while

accessing a file, if the value is set to one (1), line buffering

will be per formed while accessing a file.

A file object allows us to use, access and manipulate all the user accessible files.

**Example:**

**f = open('pyth')**#opens the file pyth for reading

**f = open('pyth', 'r')**#opens the file pyth for reading

**f = open('pyth', 'w' )**#opens the file pyth for writing

Files can also be created by calling the file object constructor , which is identical to open().

f = file('pyth') #opens pyth for reading

 f = file('pyth', 'w') #opens pyth for writing

**File modes in Python:**

r – Read mode which is used when the file is only being read.

w – Write mode which is used to edit and write new information to the file.

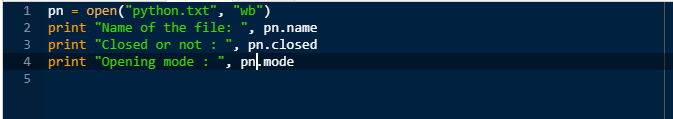
a – Appending mode, which is used to add new data to the end of the file; that is new information is automatically amended to the end.

**t**- This is the default mode. It opens in text mode

**x** - Creates a new file. If file already exists, the operation fails.

The mode character can be followed by **b**for binary data such as 'rb' or 'wb'.

A file can be opened for updates by supplying a plus(+) such as 'r+' or 'w+'.



**Output:**

Name of the file: python.txt

Closed or not : False

Opening mode : wb

#### **1.2. Read data from file**

**Syntax**

**fileObject.read()**

**read()**- This method returns the entire file as string unless an optional length parameter is given specifying the maximum number of bytes.

**readline()** - This method returns the next line of input, including the terminating newline. This methid optionally accepts a maximum line length,n. If a line length longer than n is read, the first n bytes are returned. The remaining line data is not discarded and will be returned on subsequent read operations.

**readlines()** - This method returns all the input lines as a list of strings. This method accepts a size parameter that specifies the approximate number of bytes to read before stopping. The actual number of bytes read may be larger than this depending on how much data has been buffered.

**xreadlines()** - This method returns an iterator for reading the file line by line. This method is only provided for backward compatibility because files can already be used as iterators.

https://g91.tcsion.com/per/g91/pub/2030/LX/ckeditor_assets/pictures/2030/2560/image_88508_original.png

**Output:** Read String is : Python is

# 1.3. Write data to file

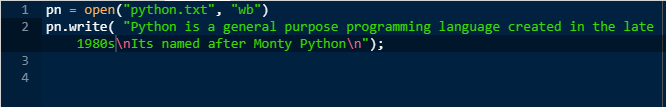
**Syntax**

**fileObject.write()**

**write()**method writes a string to a file.

**writelines()** method writes a list of strings to th file. This method does not add newline characters to the output, so the supplied list of output stringsshould already be formatted as necessary.

In both write() and writelines() method, the string can contain binary data, including embedded NULL characters.



#### **1.4. Append data to file**

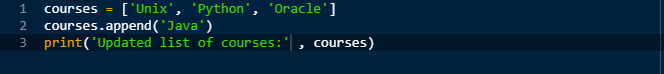
The append() method adds an item to the existing list and increases the length of the list by one. It does not return a new list; rather it modifies the original list.

**Syntax:**

**list\_name.append(item)**

The append() method takes a single item and adds it to the end of the list.

The item can be numbers, strings, another list, dictionary etc.



#### Output:

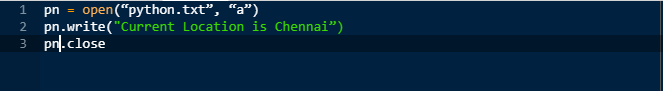
'Updated list of courses:', ['Unix', 'Python', 'Oracle', 'Java']

**1.5. Close File**

The close method of a file object removes any unwritten information and closes the file object, after which no more writing can be done. Python automatically closes a file when the reference object of a file is reassigned to another file. It is always essential to use the close method to close a file.

**Syntax**:

**fileObject.close()**

****

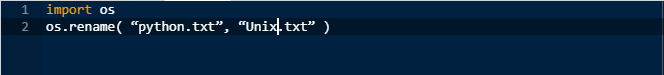
#### **1.6. Renaming and Deleting a file**

Python os module provides methods to rename and delete files.

To use these methods we need to import it first.

**Syntax:**

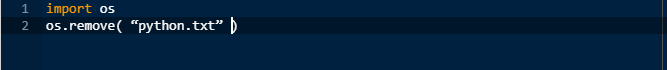
**os.rename(current\_file\_name, new\_file\_name)**



You can use the remove() method to delete files by providing the name of the file  
to be deleted as the argument.

**Syntax:**

**os.remove(file\_name)**

****

#### **1.7 File Comparison**

The filecmp module provides functions, which can be used to compare files and directories:

**Syntax:**

**cmp(file1, file2[, shallow])**

This compares the files file1 and file2 and returns True if they’re equal, False if not. By default, files that have identical attributes as returned by os.stat() are considered to be equal. If the shallow parameter is specified and is False, the contents of the two files are compared to determine equality.

**cmpfiles(dir1, dir2, common[, shallow])**

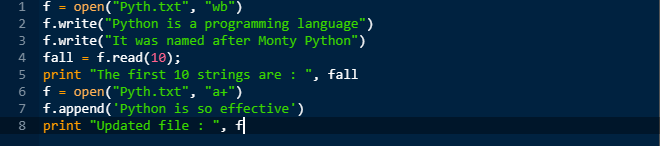
This compares the contents of the files contained in the list common in the two directories dir1 and dir2. This inturn returns a tuple containing three lists of filenames (match, mismatch, errors). match lists the files that are the same in both directories, mismatch lists the files that don’t match, and errors lists the files that could not be compared for some reason.The shallow parameter has the same meaning as for cmp().

**dircmp(dir1, dir2 [, ignore[, hide]])**

This creates a directory comparison object that can be used to perform various comparison operations on the directories dir1 and dir2. ignore is a list of filenames to ignore and hide is a list of filenames to hide.

**1.8. Sample demo case study**

Create and open a text file named Pyth.txt and insert any two lines into it. Make sure the file is opened in the right mode.Then display the first 10 characters of the file Pyth.txt. Finally append another line to the same file and display the entire contents of the file Pyth.txt.(Sample snippet is shown below)



**1. Anonymous Functions**

**1.1 Lambda**

Lambda operators or Lambda functions are used to create small functions in Python. This functions are anonymous and hence, only of one-time use.

A lambda function can have many arguments but only one expression. It cannot have a return statement. It returns a function object which can be assigned to a variable.

**Basic syntax:**

lambda argument(s) : expression

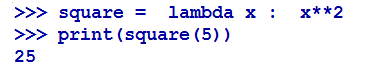
Small one liner python functions can be converted into lambda functions. Let us take an example to understand it better.

**Example:**

Suppose we have a function to find the square of a number. The function is defined in Python as shown in the below code snippet.



Now let us see how to convert this function into lambda and use it.



If we see the what type is of the square variable we see it is a function

https://g91.tcsion.com/per/g91/pub/2030/LX/ckeditor_assets/pictures/2030/2703/image_22691_original.png

## ****1.2. Map****

Maps are also anonymous functions which can be used to perform an operation on all the input items. The arguments to be provided are a function object and any number of iterables like list, dictionary etc. The function is then applicable for each item of the iterable.

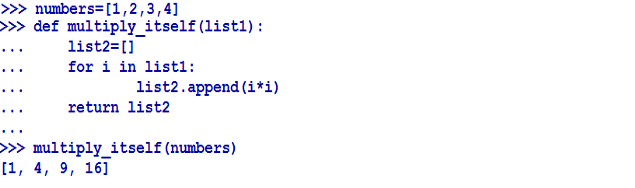
**Basic Syntax:**

map(function object, iterable 1,iterable 2,....)

**Examples:**

Let us take a function to find the product of of the elements in a list with itself.

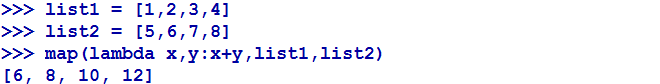
Let us define a normal function first for the requirement.



Now by using map we can do it by using lambda as shown below

https://g91.tcsion.com/per/g91/pub/2030/LX/ckeditor_assets/pictures/2030/2705/image_12834_original.png

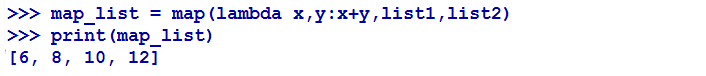
We can pass multiple sequences to map function as shown below.



Here, each element of list1 and list2 will be passed as argument one by one to the lambda function.

Neither we can access the elements of the map object with index nor we can use len() to find the length of the map object

We can force convert the map output i.e. the map object to list as shown below:



## 1.3. Filter

Like Map, Filter also takes a function object and a iterable. But here, Filter can take only one iterable as input not many which is not the case with Map.

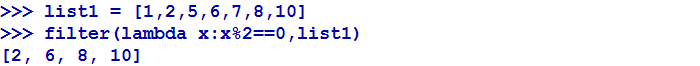
The function object returns a boolean value. This object is called for each element of the iterable and filter return only those elements for which the value of the object is true.

**Basic Syntax:**

filter(function object,iterable)

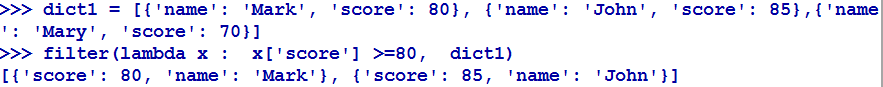
**Example:**

Let us try to understand it better with the given example



Here filter returns only those values from list for which the lambda function returns true.

Now lets try to use filter with a dictionary.



Here all the records which has a score greater than or equal to 80 are returned by the filter.

## 1.4. Reduce

Reduce also takes a function object and an iterable as input. It is useful when we want to perform s

ome operation on the list and return the result.

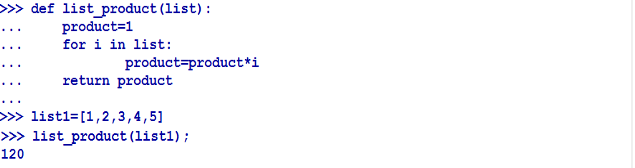
**Basic Syntax:**

reduce(function object,iterable)

**Example:**

Let us take an example of finding the product of a list of integers.

First of all, let us look at the normal function definition



If we use reduce to fulfill this requirement it can be done in a very simple way.

https://g91.tcsion.com/per/g91/pub/2030/LX/ckeditor_assets/pictures/2030/2711/image_47989_original.png

**1. Exception Handling**

Exception handling is used to handle unexpected errors in the programs. Even if the statement in your code is synatctically correct, it might throw an error at the time of execution. Hence, in that case, your program will not behave as expected. These unexpected errors which occur during runtime are called exceptions.

**Examples:**

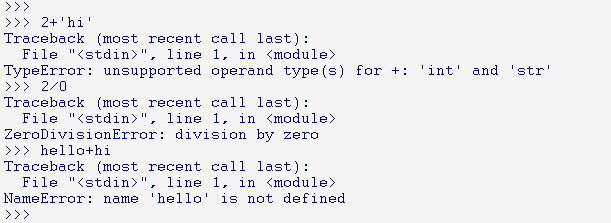
A. Transferring money to your friend though mobile Application

But sufficient funds not there in your account

B. When downloading a video / picture from Whats App to your mobile

But sufficient space is not there in your mobile storage

Let us see below what happens when such exceptions are not handled in our Python programs.



**1.1. What is an Exception?**

Exception is a runtime error due to incorrect implementation of logic, which will occur while executing an application. It is an event, which occurs during the execution of a program that disrupts the normal flow of the program's instructions.

**1.2. Issues faced due to Exception**

***1. Application, which is being run, stops abruptly***

*2.****Data inconsistency / Data loss****:*Data Loss / data inconsistency might occur in case of an exception as the application gets crashed in the middle of the data processing OR while updating DataBase, ie Partial update might happen.

*3.****Deadlock situation on common resources****:*This leads to shortage of common resources like Memory, Scanner, Database connectivity, third party components, Printer Resource shortage. When an exception occurs, application blocks the resources being used. Then the application will not allow others to use the Resources.

**For Example:**Memory usage, Connection to external resources like printer, Scanner,Database and *third party components*etc

*4.****End user may not understand the problem.***There is no clarity for the end user on the errors, like whether the exception is due to user mistake or application behaviour.As we have seen, the impact is more on an application when an exception occurs.

Solution for all these issues is “Handling the Exception “properly.

**1.3 Handling Exception**

Handling all the run time errors by conveying the user with a proper message and bringing the application back to its previous state , is called exception handling.

Developer's responsibility for effective exception handling :

During the Application development, the programmer should know about the

-Suspicious code in the logic, which might raise an exception

-What actions needs to be performed when an exception occurs

-How to make suspicious code block to communicate with the actions block upon exception.

**To implement Exception handling , we will use try and except blocks in Python:**

1. **Suspicious code block / try block –**If you have some *suspicious*code that may raise an exception, you can defend your program by placing the suspicious code in a **try**block.

Let us see some scenarios, which will cause an exception to occur and as programmers,we would need to take care of necessary measures for handling the exceptions effectively.

Some of those measures are,

* By validating the input data before performing operations like Date, String, Numeric validations.
* By validating the data before performing Arithmetic operations

For Ex: Denominator validation

* By appropriate memory management with a check on **allocation of Memory**and **Memory existence**before accessing any pointer variable.
* By Checking the memory bounds reserved before accessing the appropriate memory f

For Ex: Array index out of bounds.

* By validating the existence of Resources like DB tables, column and third party components like library before accessing the resources in code logic
* By data validations after fetch from DB/Files to check for Data Existence, Too many rows etc.

After identification of such suspicious codes, we need to keep the code in **try block**so thatwhenever an exception is raised due to the code in the **try**block control moves on to the exception handling block, i.e. **except block.**

2. **Exception Handling block / Except block:**

This block is mandatory block which will be executed whenever an exception occurs, So we need to keep all the actions to be performed **after**exception occurs and **before**the application gets closed in this block.

So this is the block, where we need to release

* All resources being used by the application like connection with third party components like printer, scanner and libraries before application gets terminated.
* We will print custom message swhich describes the error to the

user.

**Action taken by the Except block:**

* Releasing the resources blocked by the application
* Rollback the partial data updates
* Un allocating the memory, being used by the application
* Disconnect the connectivity of the third party components like Printer, Scanner, libraries and data base connectivity
* Conveying the user with meaningful message.

**3. Else block**: The else block will be executed if no expection occured in the try block.

**1.3.1. Syntax**

Here is simple syntax of t*ry....except...else*block:

*#!/usr/bin/python*

*#Suspicious code block*

*try:*

*# Logic / operations here, which could be suspicious*

*......................*

*except Exception1:*

*# If there is an exception (Exception1), then execute this block.*

*#Exception Handling block for the Exception1*

*except Exception2:*

*# If there is an exception (Exception2), then execute this block.*

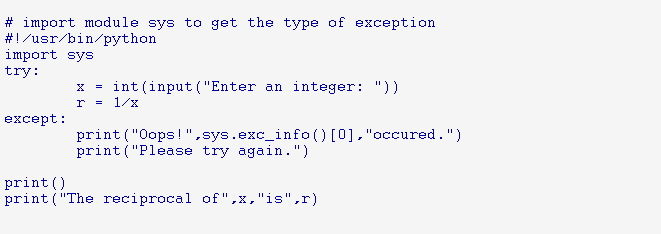
*#Exception Handling block for the Exception2*

*else:*

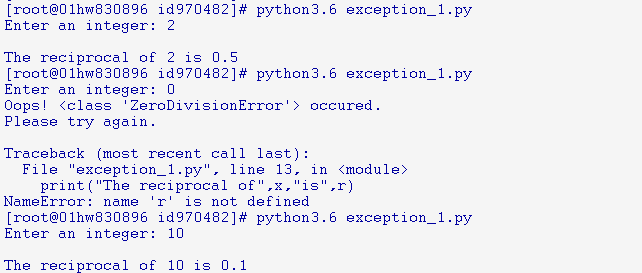
*If there is no exception then execute this block*.

**​​​​​​1.3.2 Examples**

**Example1:**



The output of the above code is shown below



**Description on Execution Flow:**

In this program, we loop until the user enters an integer that has a valid reciprocal.

The **code**that can cause exception is placed inside **try**block.

If no exception occurs, **except**block is skipped and normal flow continues.

But if any exception occurs, it is caught by the **except**block.

Here, we print the name of the exception using **ex\_info()**function inside sys module and ask the

user to try again. We can see that the values 'a' and '1.3' causes ValueError and '0' causes ZeroDivisionerror1.

**Example2:**

The below program describes to read the records from the ProductDetails.txt file, Which contain

All the products in the Store.

After reading each product it will verify that, the record is if belongs to Soap/Shampoo/Brush type,

It will Display the record for the searcher else it wont display.

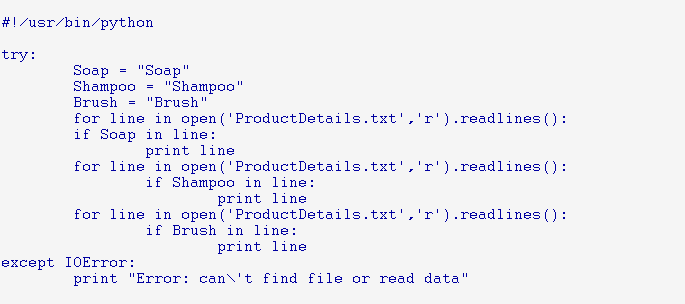
*As part of this process of execution, There could be chance that, ProductDetails.txt file might not be present or does not have read access.*

*This leads to runtime error, So By handling this error, we will address to the user saying that*

*Error: can't find file or read data,*

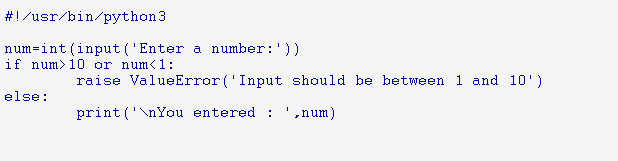
*Then user will be alerted with the mistake and check for the file or change the permission*

*So in the next run, The user would see the smooth run, without any runtime error.*

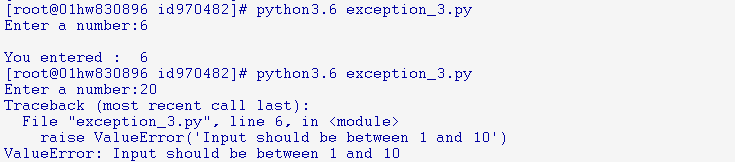
**

**1.4. Raising Exception**

The raise statement is used to explicitly force an expection to occur in the code for a particular condition. It allows you to throw an expection any time in the code.Let us take an example to understand it better.Suppose ,if we want to check if a user input is within a range( e.g. Between 1 and 10). If it is not within the range, we will raise a value error saying that input is not within range.



The output of the above code is given below:



**1.5.  References & Video Link**

<http://www.souravsengupta.com/cds2015/python/LPTHW.pdf>

<https://www.learnpython.org/>