**Syllabus**

**Introduction to Script**

* What is Script, program?
* Types of Scripts
* Difference between Script and Programming Languages
* Features and Limitation of Scripting
* Types of programming Language Paradigms

**Introduction to Python**

* What is Python?
* Why Python?
* Who Uses Python?
* Characteristics of Python
* History of Python
* What is PSF?
* Python Versions
* How to Download and Install Python
* Features and Limitations of Python
* Python Applications
* Creating Your First Python Program
* Reading Keyboard Input

**Different Modes in PYTHON**

* Execute the Script
* Interactive and Script Mode
* Python File Extensions
* Clear screen inside python
* Learn Python Main Function
* Python Comments
* Quit the Python Shell
* Shell as a Simple Calculator
* Order of operations
* Multiline Statements
* Quotations in Python
* Joining two lines
* Python Implementation Alternatives
* Sub Packages in Python

**PYTHON NEW IDEs**

* PyCharm IDE
* How to Work on PyCharm
* PyCharm Components
* Debugging process in PyCharm
* PYTHON Install Anaconda
* What is Anaconda?
* Coding Environments
* Spyder Components
* General Spyder Features
* Spyder Shortcut Keys
* Jupyter Notebook
* What is Conda? and Conda List?
* Jupyter and Kernels
* What is PIP?

**Variables in Python**

* What is Variable?
* Variables and Constants in Python
* Values and Types
* Multiple Assignment
* Python Mathematics
* Type Conversion
* Mutable Versus Immutable Objects

**String Handling**

* What is string?
* String operations
* String Functions
* Delete a string
* Python Keywords, Identifiers and Literals
* String Formatting Operator
* Structuring with indentation in Python
* Define Data Structure?

**Python Operators and Operands**

* Arithmetic, Relational Operators and Comparison Operators
* Python Assignment Operators
* Short hand Assignment Operators
* Logical Operators or Bitwise Operators
* Membership Operators
* Identity Operators
* Operator precedence
* Operators and Operands

**Python Conditional Statements**

* How to use “if condition” in conditional structures
* if statement (One-Way Decisions)
* if .. else statement (Two-way Decisions)
* if .. elif .. else statement (Multi-way)
* Nested IF Statement

**Python LOOPS**

* How to use “While Loop” and  “For Loop”
* Break statements, Continue statement, Enumerate function for For Loop and Practical Example
* How to use for loop to repeat the same statement over and again

**Learning Python Strings**

* Accessing Characters in Strings
* Various String Operators
* Some more examples
* Python String replace() Method
* Changing upper and lower case strings
* Using “join” function for the string
* Reversing String
* Split Strings

**Sequence or Collections in PYTHON**

* Strings
* Unicode Strings
* Lists
* Tuples
* xrange

**Python Lists**

* Lists are mutable
* List indices
* Traversing a list
* List operations, slices and methods
* Map, filter

**Python TUPLE**

* Advantages of Tuple over List
* Packing and Unpacking
* Comparing tuples
* Creating nested tuple
* Using tuples as keys in dictionaries
* Deleting Tuples
* Slicing of Tuple
* Tuple Membership Test
* Built-in functions with Tuple
* Dotted Charts

**Python Sets**

* How to create a set?
* Iteration Over Sets
* Python Set Methods
* Python Set Operations
* Union of sets
* Built-in Functions with Set
* Python Frozenset

**Python Dictionary**

* How to create a dictionary?
* PYTHON HASHING?
* Python Dictionary Methods
* Copying dictionary
* Updating Dictionary
* Delete Keys from the dictionary
* Dictionary items() Method
* Sorting the Dictionary
* Python Dictionary in-built Functions
* Dictionary len() Method
* Variable Types
* Python List cmp() Method
* Dictionary Str(dict)

**Python Functions**

* What is a function?
* How to define and call a function in Python
* Types of Functions
* Significance of Indentation (Space) in Python
* How Function Return Value?
* Types of Arguments in Functions
* Default Arguments and Non-Default Arguments
* Keyword Argument and Non-keyword Arguments
* Arbitrary Arguments
* Rules to define a function in Python
* Various Forms of Function Arguments
* Scope and Lifetime of variables
* Nested Functions
* Call By Value, Call by Reference
* Anonymous Functions/Lambda functions
* Passing functions to function
* map(), filter(), reduce() functions
* What is a Docstring?

**Advanced Python**

**Python Modules**

* What is a Module?
* Types of Modules
* The import Statement
* The from…import Statement
* ..import \* Statement
* Underscores in Python
* The dir( ) Function
* Creating User defined Modules
* Command line Arguments
* Python Module Search Path

**Packages in Python**

* What is a Package?
* Introduction to Packages?
* py file
* Importing module from a package
* Creating a Package
* Creating Sub Package
* Importing from Sub-Packages
* Popular Python Packages

**Python Date and Time**

* How to Use Date & DateTime Class
* How to Format Time Output
* How to use Timedelta Objects
* Calendar in Python
* datetime classes in Python
* How to Format Time Output?
* The Time Module
* Python Calendar Module
* Python Text Calendar, HTML Calendar Class
* Unix Date and Time Commands

**File Handling**

* What is a data, Information File?
* File Objects
* File Different Modes and Object Attributes
* How to create a Text Fil and Append Data to a File and Read a File
* Closing a file
* Read, read line ,read lines, write, write lines…!!
* Renaming and Deleting Files
* Directories in Python
* Working with CSV files and  CSV Module
* Handling IO Exceptions

**Python OS Module**

* Shell Script Commands
* Various OS operations in Python
* Python File System Shell Methods

**Python Exception Handling**

* Python Errors
* Common RunTime Errors in PYTHON
* Abnormal termination
* Chain of importance Of Exception
* Exception Handling
* Try … Except
* Try .. Except .. else
* Try … finally
* Argument of an Exception
* Python Custom Exceptions
* Ignore Errors
* Assertions
* UsingAssertionsEffectively

**More Advanced PYTHON**

* Python Iterators, Generators, Closures, Decorators and Python @property

**Python Class and Objects**

* Introduction to OOPs Programming
* Object Oriented Programming System
* OOPS Principles
* Define Classes
* Creating Objects
* Class variables and Instance Variables Constructors
* Basic concept of Object and Classes
* Access Modifiers
* How to define Python classes
* Python Namespace
* Self-variable in python
* Garbage Collection
* What is Inheritance? Types of Inheritance?
* How Inheritance works?
* Python Multiple Inheritance
* Overloading and Over Riding
* Polymorphism
* Abstraction
* Encapsulation
* Built-In Class Attributes

**Python Regular Expressions**

* What is Regular Expression?
* Regular Expression Syntax
* Understanding Regular Expressions
* Regular Expression Patterns
* Literal characters
* Repetition Cases
* Example of w+ and ^ Expression
* Example of \s expression in re.split function
* Using regular expression methods
* Using re.match()
* Finding Pattern in Text (re.search())
* Using re.findall for text
* Python Flags
* Methods of Regular Expressions

**Python XML Parser**

* What is XML?
* Difference between XML and HTML  and XML,  JSON, Gson
* How to Parse XML and Create XML Node
* Python vs JAVA
* XML and HTML

**Python-Data Base Communication**

* What is Database? Types of Databases?
* What is DBMS?, RDBMS?
* What is Big Data? Types of data?
* Oracle
* MySQL
* SQL server
* DB2
* Postgre SQL
* Executing the Queries
* Bind Variables
* Installing of Oracle Python Modules
* Executing DML Operations..!!

**Multi-Threading**

* What is Multi-Threading
* Threading Module
* Defining a Thread
* Thread Synchronization

**Web Scrapping**

* The components of a web page
* BeautifulSoup
* Urllib2
* HTML,CSS,JS,jQuery
* Dataframes
* PIP
* Installing External Modules Using PIP

**Unit Testing with PyUnit**

* What is Testing?
* Types of Testings and Methods?
* What is Unit Testing?
* What is PyUnit?
* Test scenarios, Test Cases, Test suites

**Introduction to Python Web Frameworks**

* Django – Design
* Advantages of Django
* MVC and MVT
* Installing Django
* Designing Web Pages
* HTML5, CSS3, AngularJS
* PYTHON Flask, Bottle, Pyramid, Falcon

**GUI Programming-Tkinter**

* Introduction
* Components and Events
* Adding Controls
* Entry Widget, Text Widget, Radio Button, Check Button
* List Boxes, Menus, ComboBox

**Data Analytics**

* Introduction to data Big Data?
* Introduction to NumPY and SciPY
* Introduction to Pandas and MatPlotLib

Introduction to script:

**1. What is script, program?**

Script:

A Script or Scripting language is a programming language designed to communicate and integrate with other programming language.

Basically all scripting languages are programming languages.

Program:

A Program or Programming language is formal language, which comprises the set of instructions which is used to produce the various kinds of output. A programming language is a computer language engineered to create a standard form of commands.

**2. Difference between scripting and programming language?**

The theoretical difference between them is scripting language does not use compilation step rather they are interpreted whereas programming language uses compilation step.

For example ‘C’ language requires compilation step whereas JavaScript, python does not require compilation step.

Scripting languages are usually slower than Programming language. Because compilation so faster than interpretation.

Also, compilers read and analyze the code only once, and report the errors collectively that the code might have, but the interpreter will read and analyze the code statements each time it meets them and halts at that very instance if there is some error.

|  |  |
| --- | --- |
| **Scripting Language** | **Programming Language** |
| It is a interpreter based language | It is a compiler based language |
| To combine existing language | Developing anything needful from scratch |
| These run inside another program | They run independent of another program |
| Design to making code fast and simple to usage | Design to give full usage of language |
| Line by line conversion | One shot conversion |
| It does not create .exe file | It creates the .exe file |
| Scripts are just piece of code | Programming is making a full code of program |
| They are easy to use and easy to write | They are complex |
| It takes less time to develop | It take more time to develop |

**3. Feature of scripting language?**

A scripting language is mainly used to automate the tasks that human could do. If you feel like you are repeatedly doing the same and thereby wasting your time, you should consider automating this task with a scripting language

Features:

* Flexible dynamic typing.

Python is a **dynamically typed** language. It doesn’t know about the type of the variable until the code is run. So declaration is of no use. What it does is, It stores that value at some memory location and then binds that variable name to that memory container. And makes the contents of the container accessible through that variable name. So the data type does not matter. As it will get to know the type of the value at run-time. Easy access to other programs

* Sophisticated pattern matching
* High level data types.
* Simple scoping rules.

**4. Limitation of scripting language?**

* Scripting languages are slower because they require an interpreter to run the code
* It is easy to change the code.
* We often have to do extra work to leverage legacy numerical libraries that are only available in Fortran or C

**5. Types of programming Languages?**

There are two types of programming languages, which can be categorized into the following ways:

a. Low level language:

* Machine language
* Assembly language

b. High level language:

* Procedural-Oriented language
* Problem-Oriented language
* Natural language

a).Low level language

* This language is the most understandable language used by computer to perform its operations. It can be further categorized into:

**i)**Machine Language

* Machine language consists of strings of binary numbers (i.e. 0s and 1s) and it is the only one language, the processor directly understands. Machine language has an Merits of very fast execution speed and efficient use of primary memory.

Merits:

* It is directly understood by the processor so has faster execution time since the programs written in this language need not to be translated.
* It doesn’t need larger memory.

Demerits:

* It is very difficult to program using machine language since all the instructions are to be represented by 0s and 1s.
* Use of this language makes programming time consuming.
* It is difficult to find error and to debug.
* It can be used by experts only.

ii) Assembly Language

* Assembly language is also known as low-level language because to design a program programmer requires detailed knowledge of hardware specification. This language uses mnemonics code (symbolic operation code like ‘ADD’ for addition) in place of 0s and 1s. The program is converted into machine code by assembler. The resulting program is referred to as an object code.

Merits:

* It is makes programming easier than machine language since it uses mnemonics code for programming.

Eg: ADD for addition, SUB for subtraction, DIV for division, etc.

* It makes programming process faster.
* Error can be identified much easily compared to machine language.
* It is easier to debug than machine language.

Demerits:

* Programs written in this language is not directly understandable by computer so translators should be used.
* It is hardware dependent language so programmers are forced to think in terms of computer’s architecture rather than to the problem being solved.
* Being machine dependent language, programs written in this language are very less or not portable.
* Programmers must know its mnemonics codes to perform any task.

b).High level language

* Instructions of this language closely resembles to human language or English like words. It uses mathematical notations to perform the task. The high level language is easier to learn. It requires less time to write and is easier to maintain the errors. The high level language is converted into machine language by one of the two different languages translator programs; interpreter or compiler.

High level language can be further categorized as:

i)Procedural-Oriented language:

* Procedural Programming is a methodology for modeling the problem being solved, by determining the steps and the order of those steps that must be followed in order to reach a desired outcome or specific program state. These languages are designed to express the logic and the procedure of a problem to be solved. It includes languages such as Pascal, COBOL, C, FORTAN, etc.

Merits:

* Because of their flexibility, procedural languages are able to solve a variety of problems.
* Programmer does not need to think in term of computer architecture which makes them focused on the problem.
* Programs written in this language are portable.

Demerits:

* It is easier but needs higher processor and larger memory.
* It needs to be translated therefore its execution time is more.

ii) Problem-Oriented language:

* It allows the users to specify what the output should be, without describing all the details of how the data should be manipulated to produce the result. This is one step ahead from 3GL. These are result oriented and include database query language.

Eg: Visual Basic, C#, PHP, etc.

objectives:

* Increase the speed of developing programs.
* Minimize user’s effort to obtain information from computer.
* Reduce errors while writing programs.

Merits:

* Programmer need not to think about the procedure of the program. So, programming is much easier.

Demerits:

* It is easier but needs higher processor and larger memory.
* It needs to be translated therefore its execution time is more.

iii) Natural language:

* Natural language are still in developing stage where we could write statements that would look like normal sentences.

Merits:

* Easy to program.
* Since, the program uses normal sentences, they are easy to understand.
* The programs designed using Natural language will have artificial intelligence (AI).
* The programs would be much more interactive and interesting.

Demerits:

* It is slower than previous generation language as it should be completely translated into binary code which is a tedious task.
* Highly advanced and expensive electronic devices are required to run programs developed in Natural language. Therefore, it is an expensive approach.

Introduction to Python

**What is Python?**

* Python is an interpreted, object-oriented, high-level programming language

Interpreted:

Python is called an interpreted language because it goes through an interpreter, which turns code you write into the language understood by your computer’s processor (Machin code).

Object oriented:

Python supports different program approach. One of the popular approach is to solve the programming problems by creating the objects. This is known as object oriented programming.

An object has two characteristics:

* Attributes
* Behavior

High level programming language:

High level languages are easier to understand and user friendly. With the help of these languages we can develop the applications that are portable across various platforms and independent of any architecture.

**Why Python?**

* Often, programmers fall in love with Python because of the increased productivity it provides. Since there is no compilation step, the edit-test-debug cycle is incredibly fast. Debugging Python programs is easy: a bug or bad input will never cause a segmentation fault. Instead, when the interpreter discovers an error, it raises an exception. When the program doesn't catch the exception, the interpreter prints a stack trace.
* It has extensive support libraries
* Portable language
* Interactive language

**Who uses Python?**

* Google makes extensive use of Python in its **web search system**, and employs Python’s creator Guido van Rossum.
* The YouTube video sharing service is largely written in Python.
* Disney uses Python in many of their creative processes.
* Mozilla uses Python to explore their extensive code base and releases tons of open source packages built in python.
* Drop box file hosting service is implemented using Python, Guido van Rossum now working here.
* The popular **Bit Torrent peer-to-peer** file sharing system is a Python program.
* **Intel, Cisco, Hewlett-Packard, Seagate, Qualcomm, and IBM** use Python for hardware testing.
* **JPMorgan Chase, UBS, Getco, and Citadel** apply Python for financial market forecasting.
* **NASA, Los Alamos, JPL**, use Python for scientific programming tasks.
* iRobot uses Python to develop **commercial robotic vacuum cleaners**.
* The NSA uses Python for **cryptography and intelligence analysis.**

**Characteristics of Python**

* Both Batch and Interactive use.
* Economy of Expression.
* Lack of declarations; **simple** scoping rules.
* Flexible **dynamic** typing.
* **Easy** access to other programs.
* Sophisticated Pattern matching.
* High-level data types.

**History of Python**

Python was conceived in the late 1980s by [Guido van Rossum](https://en.wikipedia.org/wiki/Guido_van_Rossum) at [Centrum Wiskunde & Informatica](https://en.wikipedia.org/wiki/Centrum_Wiskunde_%26_Informatica) (CWI) in the [Netherlands](https://en.wikipedia.org/wiki/Netherlands) as a successor to the [ABC language](https://en.wikipedia.org/wiki/ABC_(programming_language)) (itself inspired by [SETL](https://en.wikipedia.org/wiki/SETL)), capable of [exception handling](https://en.wikipedia.org/wiki/Exception_handling) and interfacing with the [Amoeba](https://en.wikipedia.org/wiki/Amoeba_(operating_system)) operating system.[[7]](https://en.wikipedia.org/wiki/Python_(programming_language)#cite_note-faq-created-7) Its implementation began in December 1989. Van Rossum's long influence on Python is reflected in the title given to him by the Python community: [*Benevolent Dictator For Life*](https://en.wikipedia.org/wiki/Benevolent_Dictator_For_Life) (BDFL) – a post from which he gave himself permanent vacation on July 12, 2018.

Python 2.0 was released on 16 October 2000 with many major new features, including a [cycle-detecting](https://en.wikipedia.org/wiki/Cycle_detection) [garbage collector](https://en.wikipedia.org/wiki/Garbage_collection_(computer_science)) and support for [Unicode](https://en.wikipedia.org/wiki/Unicode).

Python 3.0 was released on 3 December 2008. It was a major revision of the language that is not completely [backward-compatible](https://en.wikipedia.org/wiki/Backward_compatibility). Many of its major features were [backported](https://en.wikipedia.org/wiki/Backporting) to Python 2.6.x and 2.7.x version series. Releases of Python 3 include the 2to3 utility, which automates (at least partially) the translation of Python 2 code to Python 3.

Python 2.7's [end-of-life](https://en.wikipedia.org/wiki/End-of-life_(product)) date was initially set at 2015 then postponed to 2020 out of concern that a large body of existing code could not easily be forward-ported to Python 3. In January 2017, Google announced work on a Python 2.7 to [Go](https://en.wikipedia.org/wiki/Go_(programming_language)" \o "Go (programming language))[transcompiler](https://en.wikipedia.org/wiki/Transcompiler) to improve performance under concurrent workloads

**What is PSF?**

The Python Software Foundation (PSF) is a 501(c) (3) non-profit corporation that holds the intellectual property rights behind the Python programming language. It manage the open source licensing for Python version 2.1 and later and own and protect the trademarks associated with Python.

The mission of the Python Software Foundation is to promote, protect, and advance the Python programming language, and to support and facilitate the growth of a [diverse](https://www.python.org/psf/diversity/) and international community of Python programmers.

**Python Versions and Release dates**

Release dates for the major and minor versions

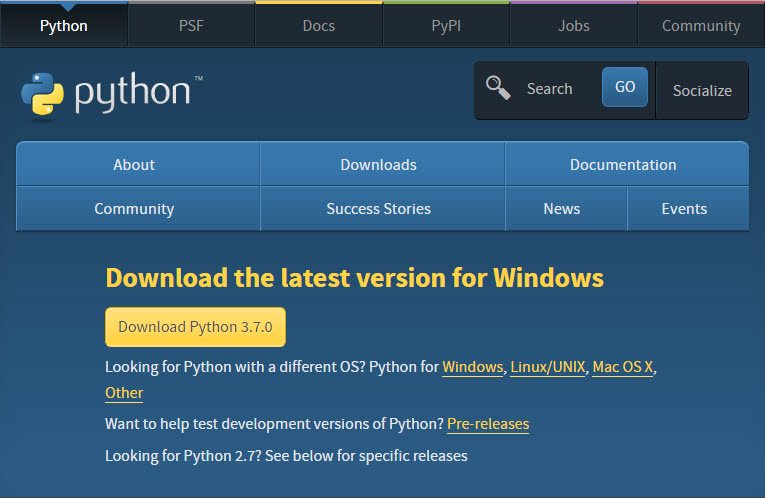
* Implementation started - December, 1989
* Internal releases at [Centrum Wiskunde & Informatica](https://en.wikipedia.org/wiki/Centrum_Wiskunde_%26_Informatica) - 1990
* Python 0.9.0 - February 20, 1991
  + Python 0.9.1 - February, 1991
  + Python 0.9.2 - Autumn, 1991
  + Python 0.9.4 - December 24, 1991
  + Python 0.9.5 - January 2, 1992
  + Python 0.9.6 - April 6, 1992
  + Python 0.9.8 - January 9, 1993
  + Python 0.9.9 - July 29, 1993
* Python 1.0 - January 1994
  + Python 1.2 - April 10, 1995
  + Python 1.3 - October 12, 1995
  + Python 1.4 - October 25, 1996
  + Python 1.5 - December 31, 1997
  + Python 1.6 - September 5, 2000
* Python 2.0 - October 16, 2000
  + Python 2.1 - April 15, 2001
  + Python 2.2 - December 21, 2001
  + Python 2.3 - July 29, 2003
  + Python 2.4 - November 30, 2004
  + Python 2.5 - September 19, 2006
  + Python 2.6 - October 1, 2008
  + Python 2.7 - July 3, 2010
* Python 3.0 - December 3, 2008
  + Python 3.1 - June 27, 2009
  + Python 3.2 - February 20, 2011
  + Python 3.3 - September 29, 2012
  + Python 3.4 - March 16, 2014
  + Python 3.5 - September 13, 2015
  + Python 3.6 - December 23, 2016
  + Python 3.7 - June 27, 2018

**How to Download and Install Python**

### Downloading

1. Click [Python Download](https://www.python.org/downloads/).

The following page will appear in your browser.



1. Click the Download Python 3.7.0(Latest version) button.

The file named python-3.7.0.exe should start downloading into your standard download folder. This file is about 30 Mb so it might take a while to download fully if you are on a slow internet connection (it took me about 10 seconds over a cable modem).

The file should appear as

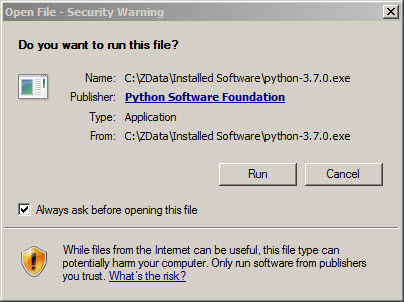
https://www.ics.uci.edu/~pattis/common/handouts/pythoneclipsejava/images/python/exefile.jpg

1. Move this file to a more permanent location, so that you can install Python (and reinstall it easily later, if necessary).
2. Feel free to explore this webpage further; if you want to just continue the installation, you can terminate the tab browsing this webpage.
3. Start the Installing instructions directly below.

### Installing

1. Double-click the icon labeling the file python-3.7.0.exe.

An Open File - Security Warning pop-up window will appear.



1. Click Run.

A Python 3.7.0 (32-bit) Setup pop-up window will appear.

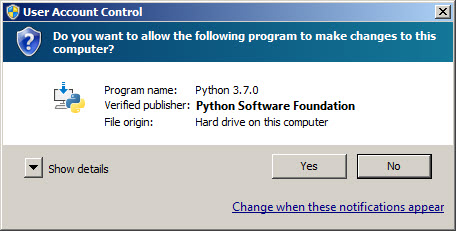


Ensure that the Install launcher for all users (recommended) and the Add Python 3.7 to PATH checkboxes at the bottom are checked.

If the Python Installer finds an earlier version of Python installed on your computer, the Install Now message may instead appear as Upgrade Now (and the checkboxes will not appear).

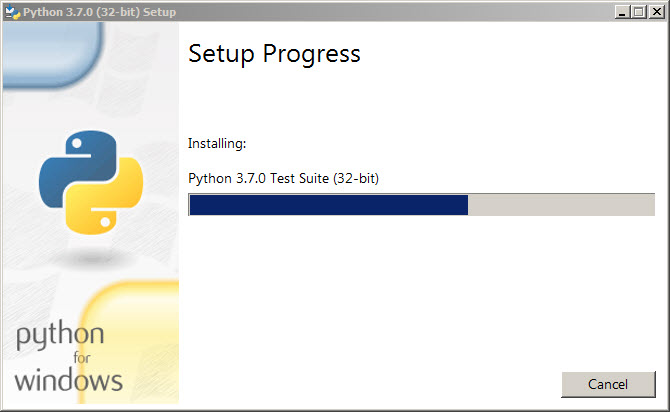
1. Highlight the Install Now (or Upgrade Now) message, and then click it.

A User Account Control pop-up window will appear, posing the question Do you want the allow the following program to make changes to this computer?

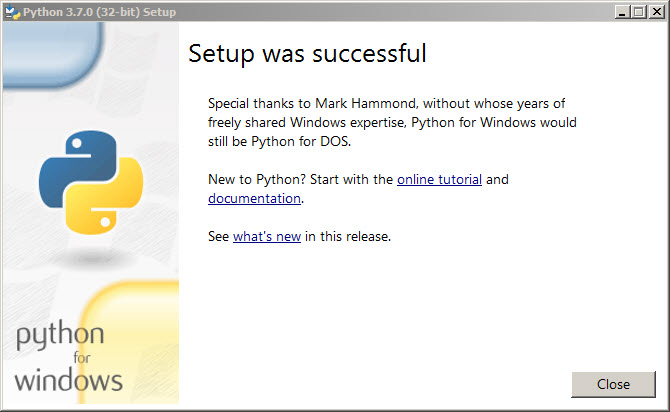


1. Click the Yes button.

A new Python 3.7.0 (32-bit) Setup pop-up window will appear with a Setup Progress message and a progress bar.



During installation, it will show the various components it is installing and move the progress bar towards completion. Soon, a new Python 3.7.0 (32-bit) Setup pop-up window will appear with a Setup was successfully message.



1. Click the Close button.

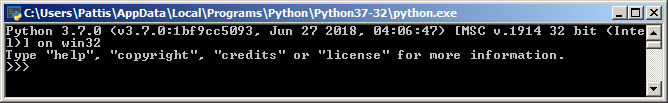
Python should now be installed.

### Verifying

To try to verify installation,

1. Navigate to the directory C:\Users\Pattis\AppData\Local\Programs\Python\Python37-32 (or to whatever directory Python was installed: see the pop-up window for installing step 3).
2. Double-click the icon/file python.exe.

The following pop-up window will appear.



A pop-up window with the title C:\Users\Pattis\AppData\Local\Programs\Python\Python37-32 appears, and inside the window; on the first line is the text Python 3.7.0 ... (notice that it should also say 32 bit). Inside the window, at the bottom left, is the prompt >>>: type exit () to this prompt and press enter to terminate Python.

You should keep the file python-3.7.0.exe somewhere on your computer in case you need to reinstall Python (not likely necessary).

You may now follow the instructions to download and install Java (you should have already installed Java, but if you haven't, it is OK to do so now, so long as you install both Python and Java before you install Eclipse), and then follows the instruction to download and install the Eclipse IDE. Note: you need to download/install Java even if you are using Eclipse only for Python)

**Features and limitations of Python:**

Features

* Object Oriented Programming-driven.

Python supports the object oriented programming approach.

* A large number of resources are available for Python.

Python has a huge set of libraries.

* Straightforward and speedy
* Portability and interactivity

Python is a portable language and architecture independent. We can run the program on the various platforms.

* Ease of use and read

Most Python programmers would agree that the biggest advantage of Python is that it is easy to pick up. Ease of use and easy readability are more than just a convenience

Limitations:

* Speed

 Since it is an interpreted language, Python can be slower than other compiled languages

### Lack of mobile computing and browsers

### Python is strong in desktop and server platforms, but weak in mobile platforms. There have only been a handful of smartphone apps developed using Python

### Design restrictions

Even the biggest fans of Python would agree to certain design restrictions in the language

### because it is dynamically typed. This requires more testing and errors to turn up only during runtime

### Python is not a good choice for memory intensive tasks.

### It's near impossible to build a high-graphic 3D game using Python.

### Has limitations with database access

### Python is not good for multi-processor/multi-core work.

**Python Applications**

* GUI-Based Desktop Applications

Python has simple syntax, modular architecture, rich text processing tools and the ability to work on multiple operating systems which make it a desirable choice for developing desktop-based applications

### Web Frameworks and Web Applications:

Python has been used to create a variety of web-frameworks including CherryPy, Django, TurboGears, Bottle, Flask etc. These frameworks provide standard libraries and modules which simplify tasks related to content management, interaction with database and interfacing with different internet protocols such as HTTP, SMTP, XML-RPC, FTP and POP

### Enterprise and Business Applications:

With features that include special libraries, extensibility, scalability and easily readable syntax, Python is a suitable coding language for customizing larger applications. Reddit, which was originally written in Common Lips, was rewritten in Python in 2005. Python also contributed in a large part to functionality in YouTube.

### Operating Systems:

Python is often an integral part of Linux distributions. For instance, Ubuntu’s Ubiquity Installer, and Fedora’s and Red Hat Enterprise Linux’s Anaconda Installer are written in Python. Gentoo Linux makes use of Python for Portage, its package management system.

### Language Development:

Python’s design and module architecture has influenced development of numerous languages. Boo language uses an object model, syntax and indentation, similar to Python. Further, syntax of languages like Apple’s Swift, CoffeeScript, Cobra, and OCaml all share similarity with Python.

**Creating Your First Python Program**

File name is First.py

File.py

print “Hello world”

**Reading Keyboard Input**

input() is used to read from the users.

Ex:

readdata=input (“Enter any number”)

print “Read data is”+readdata

Different Modes in PYTHON

**Execute the Script**

To execute the script on the command line using “python filename.py”

**Interactive and Script Mode**

Python has two basic modes: script and interactive

Script mode:

The normal mode is the mode where the scripted and finished .py files are run in the Python interpreter.

Interactive mode:

Interactive mode is a command line shell which gives immediate feedback for each statement, while running previously fed statements in active memory. As new lines are fed into the interpreter, the fed program is evaluated both in part and in whole.

Interactive mode is a good way to play around and try variations on syntax.

**Python File Extensions**

The python file extension is .py

**Clear screen inside python**

Most of the time, while working with python interactive shell/terminal (not a console), we end up with a messy output and want to clear the screen for some reason.  
In an interactive shell/terminal, we can simply use

Ctrl+l

But, what if we want to clear the screen while running a python script.  
Unfortunately, there’s no built-in keyword or function/method to clear the screen. So, we do it on our own.

**Python Main Function**

When the Python interpreter reads a source file, it executes all of the code found in it.

Before executing the code, it will define a few special variables. For example, if the Python interpreter is running that module (the source file) as the main program, it sets the special \_\_name\_\_variable to have a value "\_\_main\_\_". If this file is being imported from another module, \_\_name\_\_will be set to the module's name.

Ex:

def main():

print “This is hello world”

if \_\_name\_\_ == “\_\_main\_\_”:

main ()

**Python Comments**

Comments are the useful information that the developers provide to make the reader understand the source code. It improves the readability of the whole program.

There are two types of comment in Python:

**Single line comments:**

Python single line comment starts with hashtag symbol with no white spaces (#) and lasts till the end of the line. If the comment exceeds one line then put a hashtag on the next line and continue the comment. Python’s single line comments are proved useful for supplying short explanations for variables

Ex:

Main.py

#This is single line comment

Multi line string as comment:

Python multi-line comment is a piece of text enclosed in a delimiter (""") on each end of the comment. Again there should be no white space between delimiter ("""). They are useful when the comment text does not fit into one line; therefore needs to span across lines. Multi-line comments or paragraphs serve as documentation for others reading your code

Ex:

"""

This would be a multiline comment in Python that spans several lines and describes geeksforgeeks. A Computer Science portal for geeks. It contains well written, well thought and well-explained computer science and programming articles, quizzes and more.  …"""

'''This article on geeksforgeeks gives you a perfect example of multi-line comments'''

**Quotations in Python**

We have single quotes and double quotes. As far as language syntax is concerned, there is no difference in single or double quoted string. Both representations can be used interchangeably

**Joining two lines**

We can concatenate the two lines by using “+” operator.

Ex:

Line1=”this is vidyanath”

Line2=”Nalla”

Line3=Line1+” ”+Line2

print Line3

output:

“this is vidyanath Nalla”

**Python Implementation Alternatives**

A number of alternative implementations are available as well, namely

* [IronPython](http://ironpython.net/) (Python running on .NET)
* [Jython](http://www.jython.org/) (Python running on the Java Virtual Machine)
* [PyPy](http://pypy.org/) (A [fast](http://speed.pypy.org/) python implementation with a JIT compiler)
* [Stackless Python](http://www.stackless.com/) (Branch of CPython supporting microthreads)
* [MicroPython](http://micropython.org/) (Python running on micro controllers)

**Sub Packages in Python**

Packages can contain nested **subpackages** to arbitrary depth. We can define the subpackages in the packages.

/package

|-subpackage1

| |-\_\_init\_\_.py

| |-src

| | |-\_\_init\_\_.py

| | |-my\_program.py

| | |-functions.py

| | |-...

|-subpackage2

| |-web-server.py

| |-API

| | |-\_\_init\_\_.py

| | |-REST.py

| | |-...

In the above structure, we can call any sub package file as package.subpackage1.functions.py

Variables in Python

**What is Variable?**

Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory. Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory.

[Python](https://www.geeksforgeeks.org/python-programming-language/) is not “statically typed”, but “dynamically typed”. We do not need to declare variables before using them, or declare their type. A variable is created the moment we first assign a value to it.

Each variable must have a unique name called identifier.

Ex:

# An integer assignment

age = 45

# A floating point

salary = 1456.8

# A string

name = "John

**Variables and Constants in Python**

Variables are three types.

i) Global Variables

ii) Local variables

iii) Class or Static variables

i) Global Variables:

Global variables are the one that are defined and declared outside a function and we need to use them inside a function.

Ex:

# This function uses global variable s

def f ():

print s

# Global scope

s = "I love Geeksforgeeks"

f()

If a variable with same name is defined inside the scope of function as well then it will print the value given inside the function only and not the global value.

Ex:

# This function has a variable with name same as s.

def f():

#Local scope

s = "Me too."

print s

# Global scope

s = "I love Geeksforgeeks"

f()

print s

ii) Local Variables:

A variable declared inside the function's body or in the local scope is known as local variable.

Ex:

def foo ():

y = "local"

print(y)

foo()

iii) Class or Static variables:

Class or static variables are shared by all objects. Elements outside the \_\_init\_\_ method are static elements, it means, they belong to the class.

Elements inside the \_\_init\_\_ method are elements of the object (self), they don't belong to the class

Ex:

class CSStudent:

    stream = 'cse'                  # Class Variable

    def \_\_init\_\_(self,name,roll):

        self.name = name          #Instance variable

        self.roll = roll            #Instance variable

**Constants**

A constant is a type of variable whose value cannot be changed.  It is helpful to think of constants as containers that hold information which cannot be changed later.

Non technically, you can think of constant as a bag to store some books and those books cannot be replaced once placed inside the bag.

Normally in Python, constants are capitalized which helps the programmer know it's a constant.

Ex. MY\_CONSTANT = "Whatever"

class Foo (object):

CONST\_NAME = "Name"

**Value and Types**

**value**

A **value** is one of the fundamental things like a word or a number that a program manipulates. We often refer to these values as **objects** and we will use the words value and object interchangeably.

Ex:

A=6

String=”vidyanath”

6,”vidyanath” are values.

**Types**

We have different data types.

* Integer

Ex: v=6

* Floating Point numbers

Ex: v=6.567

* Complex numbers

Ex: V=6+i9

* Strings

Ex: a=”this”

* Boolean

Ex: a =True

B=False

**Multiple Assignment**

* Multiple assignment is possible in Python.
* Multiple assignment is also known as ‘**tuple unpacking’** or ‘**iterable unpacking’.**
* Python’s multiple assignment looks like this:

Ex: x,y=10,20

* We can assign two values to two variables in a single line.
* Multiple assignment is often called “tuple unpacking” because it’s frequently used with tuples.

Ex: x,y=(10,20)

* We can use multiple assignment with any iterable, not just tuples

Ex:

x,y=[10,20]

* Multiple assignment is actually fairly strict when it comes to unpacking the iterable we give to it.

Ex:

x,y=(10,20,30)

If we try to unpack a larger iterable into a smaller number of variables, we’ll get an error:

ValueError: too many values to unpack

**Operators and Operands**

# Basic Operators in Python

* 1. Arithmetic operators
  2. **Relational Operators**
  3. **Logical operators**
  4. **Bitwise operators**
  5. **Assignment operators**
  6. **Special operators**

**Python Mathematics:**

Python offers modules like math and random to carry out different mathematics like trigonometry, logarithms, probability and statistics, etc.

Ex:

import math

# Output: 3.141592653589793

print(math.pi)

# Output: -1.0

print(math.cos(math.pi))

# Output: 22026.465794806718

print(math.exp(10))

# Output: 3.0

print(math.log10(1000))

# Output: 1.1752011936438014

print(math.sinh(1))

# Output: 720

print(math.factorial(6))

import random

# Output: 16

print(random.randrange(10,20))

x = ['a', 'b', 'c', 'd', 'e']

# Get random choice

print(random.choice(x))

# Shuffle x

random.shuffle(x)

# Print the shuffled x

print(x)

# Print random element

print(random.random())

**Type Conversion**

Python defines type conversion functions to directly convert one data type to another which is useful in day to day and competitive programming

**1. int (a, base)**:

This function converts**any data type to integer**. ‘Base’ specifies the **base in which string is**if data type is string.

# initializing string

s = "10010"

# Printing string converting to int base 2

c = int(s, 2)

print ("After converting to integer base 2: ", end="")

print (c)

**2. float ()**:

This function is used to convert **any data type to a floating point number**

# Printing string converting to float

e = float(s)

print ("After converting to float : ", end="")

print (e)

**3. ord ():**

This function is used to convert a **character to integer.**

**4. hex ():**

This function is to convert **integer to hexadecimal string**

**5. oct ():**

This function is to convert **integer to octal string.**

# Python code to demonstrate Type conversion

# using ord (), hex (), oct ()

# initializing integer

s = '4'

# printing character converting to integer

c = ord(s)

print ("After converting character to integer: ",end="")

print (c)

# printing integer converting to hexadecimal string

c = hex(56)

print ("After converting 56 to hexadecimal string: ",end="")

print (c)

# printing integer converting to octal string

c = oct (56)

print ("After converting 56 to octal string: ", end="")

print (c)

6. tuple () :

This function is used to convert to a tuple.

7. set ():

This function returns the type after converting to set.

8. list ():

This function is used to convert any data type to a list type.

# Python code to demonstrate Type conversion

# using tuple (), set (), list ()

# initializing string

s = 'geeks'

# Printing string converting to tuple

c = tuple(s)

print ("After converting string to tuple: ", end="")

print (c)

# printing string converting to set

c = set(s)

print ("After converting string to set : "end="")

print (c)

# Printing string converting to list

c = list(s)

print ("After converting string to list: “end="")

print (c)

9. dict ():

This function is used to convert a tuple of order (key, value) into a dictionary.

10. str ():

Used to convert integer into a string.

11. complex (real, imag):

This functionconverts real numbers to complex (real, imag) number.

# using dict (), complex (), str ()

# initializing integers

a = 1

b = 2

# initializing tuple

tup = (('a', 1) ,('f', 2), ('g', 3))

# printing integer converting to complex number

c = complex (1, 2)

print ("After converting integer to complex number: “end="")

print (c)

# Printing integer converting to string

c = str (a)

print ("After converting integer to string: ", end="")

print (c)

# printing tuple converting to expression dictionary

c = dict (tup)

print ("After converting tuple to dictionary: ", end="")

print (c)

**Mutable Versus Immutable Objects**

Mutable object:

Object that can be changed after creating it.

Immutable object:

Object that cannot be changed after creating it.

In python will try to change the value of the immutable object it will give the new object.

Mutable Objects

Here are the list objects in python that are of mutable type:

1. list
2. Dictionary
3. Set
4. byte array
5. user defined classes

Immutable Objects

Here are the list objects in python that are of immutable type:

1. int
2. float
3. decimal
4. complex
5. bool
6. string
7. tuple
8. range
9. frozenset
10. bytes

String Handling

**What is string?**

A string is a sequence of characters. A character is simply a symbol. For example, the English language has 26 characters.

Computers do not deal with characters, they deal with numbers (binary). Even though you may see characters on your screen, internally it is stored and manipulated as a combination of 0's and 1's.

This conversion of character to a number is called encoding, and the reverse process is decoding. ASCII and Unicode are some of the popular encoding used.

In Python, string is a sequence of Unicode character. Unicode was introduced to include every character in all languages and bring uniformity in encoding.

**String operations and indices**

String operations

There are many operations that can be performed with string which makes it one of the most used [datatypes in Python](https://www.programiz.com/python-programming/variables-datatypes).

### Concatenation of Two or More Strings

Joining of two or more strings into a single one is called concatenation.

The **+** operator does this in Python. Simply writing two string literals together also concatenates them.

The **\*** operator can be used to repeat the string for a given number of times.

Ex:

str1 = 'Hello'

str2 ='World!'

# using ‘+’

print ('str1 + str2 = ', str1 + str2) #’Hello World’

# using \*

print ('str1 \* 3 =', str1 \* 3) #HelloHelloHello

### Iterating Through String

Using [for loop](https://www.programiz.com/python-programming/for-loop) we can iterate through a string. Here is an example to count the number of 'l' in a string.

Ex:

count = 0

for letter in 'Hello World':

if (letter == 'l'):

count += 1

print(count, ‘letters found')

### String Membership Test

We can test if a sub string exists within a string or not, using the keyword in.

Ex:

‘a’ in ‘program’

True

### Built-in functions to Work with Strings

Various built-in functions that work with sequence, works with string as well.

Some of the commonly used ones are enumerate () and len(). The enumerate () function returns an enumerate object. It contains the index and value of all the items in the string as pairs. This can be useful for iteration.

Similarly, len () returns the length (number of characters) of the string.

Ex:

str = 'cold'

# enumerate ()

list\_enumerate = list (enumerate (str))

print (‘list (enumerate (str) = ', list\_enumerate)

#character count

print (‘len (str) = ', len(str))

**String Functions**

Python has quite a few methods that string objects can call to perform frequency occurring task (related to string). For example, if you want to capitalize the first letter of a string, you can use capitalize () method.

| Method | Description |
| --- | --- |
| [Python String capitalize()](https://www.programiz.com/python-programming/methods/string/capitalize) | Converts first character to Capital Letter |
| [Python String center()](https://www.programiz.com/python-programming/methods/string/center) | Pads string with specified character |
| [Python String casefold()](https://www.programiz.com/python-programming/methods/string/casefold) | converts to casefolded strings |
| [Python String count()](https://www.programiz.com/python-programming/methods/string/count) | returns occurrences of substring in string |
| [Python String endswith()](https://www.programiz.com/python-programming/methods/string/endswith) | Checks if String Ends with the Specified Suffix |
| [Python String expandtabs()](https://www.programiz.com/python-programming/methods/string/expandtabs) | Replaces Tab character With Spaces |
| [Python String encode()](https://www.programiz.com/python-programming/methods/string/encode) | returns encoded string of given string |
| [Python String find()](https://www.programiz.com/python-programming/methods/string/find) | Returns the index of first occurrence of substring |
| [Python String format()](https://www.programiz.com/python-programming/methods/string/format) | formats string into nicer output |
| [Python String index()](https://www.programiz.com/python-programming/methods/string/index) | Returns Index of Substring |
| [Python String isalnum()](https://www.programiz.com/python-programming/methods/string/isalnum) | Checks Alphanumeric Character |
| [Python String isalpha()](https://www.programiz.com/python-programming/methods/string/isalpha) | Checks if All Characters are Alphabets |
| [Python String isdecimal()](https://www.programiz.com/python-programming/methods/string/isdecimal) | Checks Decimal Characters |
| [Python String isdigit()](https://www.programiz.com/python-programming/methods/string/isdigit) | Checks Digit Characters |
| [Python String isidentifier()](https://www.programiz.com/python-programming/methods/string/isidentifier) | Checks for Valid Identifier |
| [Python String islower()](https://www.programiz.com/python-programming/methods/string/islower) | Checks if all Alphabets in a String are Lowercase |
| [Python String isnumeric()](https://www.programiz.com/python-programming/methods/string/isnumeric) | Checks Numeric Characters |
| [Python String isprintable()](https://www.programiz.com/python-programming/methods/string/isprintable) | Checks Printable Character |
| [Python String isspace()](https://www.programiz.com/python-programming/methods/string/isspace) | Checks Whitespace Characters |
| [Python String istitle()](https://www.programiz.com/python-programming/methods/string/istitle) | Checks for Titlecased String |
| [Python String isupper()](https://www.programiz.com/python-programming/methods/string/isupper) | returns if all characters are uppercase characters |
| [Python String join()](https://www.programiz.com/python-programming/methods/string/join) | Returns a Concatenated String |
| [Python String ljust()](https://www.programiz.com/python-programming/methods/string/ljust) | returns left-justified string of given width |
| [Python String rjust()](https://www.programiz.com/python-programming/methods/string/rjust) | returns right-justified string of given width |
| [Python String lower()](https://www.programiz.com/python-programming/methods/string/lower) | returns lowercased string |
| [Python String upper()](https://www.programiz.com/python-programming/methods/string/upper) | returns uppercased string |
| [Python String swapcase()](https://www.programiz.com/python-programming/methods/string/swapcase) | swap uppercase characters to lowercase; vice versa |
| [Python String lstrip()](https://www.programiz.com/python-programming/methods/string/lstrip) | Removes Leading Characters |
| [Python String rstrip()](https://www.programiz.com/python-programming/methods/string/rstrip) | Removes Trailing Characters |
| [Python String strip()](https://www.programiz.com/python-programming/methods/string/strip) | Removes Both Leading and Trailing Characters |
| [Python String partition()](https://www.programiz.com/python-programming/methods/string/partition) | Returns a Tuple |
| [Python String maketrans()](https://www.programiz.com/python-programming/methods/string/maketrans) | returns a translation table |
| [Python String rpartition()](https://www.programiz.com/python-programming/methods/string/rpartition) | Returns a Tuple |
| [Python String translate()](https://www.programiz.com/python-programming/methods/string/translate) | returns mapped charactered string |
| [Python String replace()](https://www.programiz.com/python-programming/methods/string/replace) | Replaces Substring Inside |
| [Python String rfind()](https://www.programiz.com/python-programming/methods/string/rfind) | Returns the Highest Index of Substring |
| [Python String rindex()](https://www.programiz.com/python-programming/methods/string/rindex) | Returns Highest Index of Substring |
| [Python String split()](https://www.programiz.com/python-programming/methods/string/split) | Splits String from Left |
| [Python String rsplit()](https://www.programiz.com/python-programming/methods/string/rsplit) | Splits String From Right |
| [Python String splitlines()](https://www.programiz.com/python-programming/methods/string/splitlines) | Splits String at Line Boundaries |
| [Python String startswith()](https://www.programiz.com/python-programming/methods/string/startswith) | Checks if String Starts with the Specified String |
| [Python String title()](https://www.programiz.com/python-programming/methods/string/title) | Returns a Title Cased String |
| [Python String zfill()](https://www.programiz.com/python-programming/methods/string/zfill) | Returns a Copy of The String Padded With Zeros |
| [Python String format\_map()](https://www.programiz.com/python-programming/methods/string/format_map) | Formats the String Using Dictionary |
| [Python any()](https://www.programiz.com/python-programming/methods/built-in/any) | Checks if any Element of an Iterable is True |
| [Python all()](https://www.programiz.com/python-programming/methods/built-in/all) | returns true when all elements in iterable is true |
| [Python ascii()](https://www.programiz.com/python-programming/methods/built-in/ascii) | Returns String Containing Printable Representation |
| [Python bool()](https://www.programiz.com/python-programming/methods/built-in/bool) | Converts a Value to Boolean |
| [Python bytearray()](https://www.programiz.com/python-programming/methods/built-in/bytearray) | returns array of given byte size |
| [Python bytes()](https://www.programiz.com/python-programming/methods/built-in/bytes) | returns immutable bytes object |
| [Python compile()](https://www.programiz.com/python-programming/methods/built-in/compile) | Returns a Python code object |
| [Python complex()](https://www.programiz.com/python-programming/methods/built-in/complex) | Creates a Complex Number |
| [Python enumerate()](https://www.programiz.com/python-programming/methods/built-in/enumerate) | Returns an Enumerate Object |
| [Python filter()](https://www.programiz.com/python-programming/methods/built-in/filter) | constructs iterator from elements which are true |
| [Python float()](https://www.programiz.com/python-programming/methods/built-in/float) | returns floating point number from number, string |
| [Python input()](https://www.programiz.com/python-programming/methods/built-in/input) | reads and returns a line of string |
| [Python int()](https://www.programiz.com/python-programming/methods/built-in/int) | returns integer from a number or string |
| [Python iter()](https://www.programiz.com/python-programming/methods/built-in/iter) | returns iterator for an object |
| [Python len()](https://www.programiz.com/python-programming/methods/built-in/len) | Returns Length of an Object |
| [Python max()](https://www.programiz.com/python-programming/methods/built-in/max) | returns largest element |
| [Python min()](https://www.programiz.com/python-programming/methods/built-in/min) | returns smallest element |
| [Python map()](https://www.programiz.com/python-programming/methods/built-in/map) | Applies Function and Returns a List |
| [Python ord()](https://www.programiz.com/python-programming/methods/built-in/ord) | returns Unicode code point for Unicode character |
| [Python reversed()](https://www.programiz.com/python-programming/methods/built-in/reversed) | returns reversed iterator of a sequence |
| [Python slice()](https://www.programiz.com/python-programming/methods/built-in/slice) | creates a slice object specified by range() |
| [Python sorted()](https://www.programiz.com/python-programming/methods/built-in/sorted) | returns sorted list from a given iterable |
| [Python sum()](https://www.programiz.com/python-programming/methods/built-in/sum) | Add items of an Iterable |
| [Python zip()](https://www.programiz.com/python-programming/methods/built-in/zip) | Returns an Iterator of Tuples |

**Delete a string**

In Python, strings are immutable, so you have to create a new string. You have a few options of how to create the new string. If you want to remove the 'M' wherever it appears:

newstr = oldstr.replace ("M", "")

If you want to remove the central character:

midlen = len (oldstr)/2

newstr = oldstr[:midlen] + oldstr[midlen+1:]

For deleting a string:

For deleting a string we are using ‘del’.

Ex:

String=”vidyanath”

del String

**Python Keywords, identifiers and literals**

Keywords:

Keywords are the reserved words in Python.

We cannot use a keyword as [variable name](https://www.programiz.com/python-programming/variables-datatypes), [function](https://www.programiz.com/python-programming/function) name or any other identifier. They are used to define the syntax and structure of the Python language.

In Python, keywords are case sensitive. There are 33 keywords in Python. This number can vary slightly in course of time.

All the keywords except True, False and None are in lowercase and they must be written as it is. The list of all the keywords are given below.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Keywords in Python programming language | | | | |
| False | class | finally | is | return |
| None | continue | for | lambda | try |
| True | def | from | nonlocal | while |
| and | del | global | not | with |
| as | elif | if | or | yield |
| assert | else | import | pass |  |
| break | except | in | raise |  |

Identifiers:

Identifier is the name given to entities like class, functions, variables etc. in Python. It helps differentiating one entity from another.

Rules for writing identifiers

1. Identifiers can be a combination of letters in lowercase (a to z) or uppercase (A to Z) or digits (0 to 9) or an underscore (\_). Names like myClass, var\_1 and print\_this\_to\_screen, all are valid example.
2. An identifier cannot start with a digit. 1variable is invalid, but variable1 is perfectly fine.
3. Keywords cannot be used as identifiers.
4. We cannot use special symbols like! @, #, $, % etc. in our identifier.
5. Identifier can be of any length.

Literals:

Literals can be defined as a data that is given in a variable or constant.

Python support the following literals:

String literals:

String literals can be formed by enclosing a text in the quotes. We can use both single as well as double quotes for a String.

Ex:

"Aman" , '12345'

Numeric literals:

Numeric Literals are immutable. Numeric literals can belong to following four different numerical types

|  |  |  |  |
| --- | --- | --- | --- |
| **Int(signed integers)** | **Long(long integers)** | **float(floating point)** | **Complex(complex)** |
| Numbers( can be both positive and negative) with no fractional part.eg: 100 | Integers of unlimited size followed by lowercase or uppercase L eg: 87032845L | Real numbers with both integer and fractional part eg: -26.2 | In the form of a+bj where a forms the real part and b forms the imaginary par |

Boolean literals:

A Boolean literal can have any of the two values: True or False.

Special literals.

Python contains one special literal i.e., None.

None is used to specify to that field that is not created. It is also used for end of lists in Python.

**String Formatting Operator**

Python uses C-style string formatting to create new, formatted strings. The "%" operator is used to format a set of variables enclosed in a "tuple" (a fixed size list), together with a format string, which contains normal text together with "argument specifiers", special symbols like "%s" and "%d".

Ex:

# This prints out "Hello, John!"

name = "John"

print ("Hello, %s!" % name)

**Structuring with indentation**

Python uses a different principle. Python programs get structured through indentation, i.e. code blocks are defined by their indentation.  In the case of Python it's a language requirement not a matter of style. This principle makes it easier to read and understand other people's Python code.

All statements with the same distance to the right belong to the same block of code, i.e. the statements within a block line up vertically. The block ends at a line less indented or the end of the file. If a block has to be more deeply nested, it is simply indented further to the right.

Loops and Conditional statements end with a colon ":" - the same is true for functions and other structures introducing blocks. So, we should have said Python structures by colons and indentation.

An important concept is that there are no opening and closing braces to mark the start and end of a block. There are only line indentation to mark the blocks of code which is highly enforced.

There are no braces to indicate blocks of code for class and function definitions or flow control. Blocks of code are denoted by line indentation, which is rigidly enforced.

The number of spaces in the indentation is variable, but all statements within the block must be indented the same amount.

Ex:

if True:

print "True"

else:

print "False"

**Define Data Structure?**

Data Structures are structureswhich can hold some data together. In other words, they are used to store a collection of related data.

There are four built-in data structures in Python - list, tuple, dictionary and set

Python Operators and Operands

**1. Arithmetic operators:**

Arithmetic operators are used to perform mathematical operations like addition, subtraction, multiplication and division

|  |  |  |
| --- | --- | --- |
| **OPERATOR** | **DESCRIPTION** | **SYNTAX** |
| + | Addition: adds two operands | x + y |
| - | Subtraction: subtracts two operands | x - y |
| \* | Multiplication: multiplies two operands | x \* y |
| / | Division (float): divides the first operand by the second | x / y |
| // | Division (floor): divides the first operand by the second | x // y |
| % | Modulus: returns the remainder when first operand is divided by the second | x % y |

# Examples of Arithmetic Operator

a = 9

b = 4

# Addition of numbers

add = a + b

# Subtraction of numbers

sub = a - b

# Multiplication of number

mul = a \* b

# Division (float) of number

div1 = a / b

# Division (floor) of number

div2 = a // b

# modulo of both number

mod = a % b

**2 Relational Operators**

Relational operators compares the values. It either returns **True** or **False** according to the condition.

|  |  |  |
| --- | --- | --- |
| **OPERATOR** | **DESCRIPTION** | **SYNTAX** |
| > | Greater than: True if left operand is greater than the right | x > y |
| < | Less than: True if left operand is less than the right | x < y |
| == | Equal to: True if both operands are equal | x == y |
| != | Not equal to - True if operands are not equal | x != y |
| >= | Greater than or equal to: True if left operand is greater than or equal to the right | x >= y |
| <= | Less than or equal to: True if left operand is less than or equal to the right | x <= y |

# Examples of Relational Operators

a = 13

b = 33

# a > b is False

print (a > b)

# a < b is True

print (a < b)

# a == b is False

print(a == b)

# a != b is True

print(a != b)

# a >= b is False

print(a >= b)

# a <= b is True

print(a <= b)

**3.Logical operators:**

Logical operators perform Logical AND, Logical OR and Logical NOT operations.

|  |  |  |
| --- | --- | --- |
| **OPERATOR** | **DESCRIPTION** | **SYNTAX** |
| and | Logical AND: True if both the operands are true | x and y |
| or | Logical OR: True if either of the operands is true | x or y |
| not | Logical NOT: True if operand is false | not x |

# Examples of Logical Operator

a = True

b = False

# Print a and b is False

print(a and b)

# Print a or b is True

print (a or b)

# Print not a is False

print (not a)

**4. Bitwise operators:**

Bitwise operators acts on bits and performs bit by bit operation.

|  |  |  |
| --- | --- | --- |
| **OPERATOR** | **DESCRIPTION** | **SYNTAX** |
| & | Bitwise AND | x & y |
| | | Bitwise OR | x | y |
| ~ | Bitwise NOT | ~x |
| ^ | Bitwise XOR | x ^ y |
| >> | Bitwise right shift | x>> |
| << | Bitwise left shift | x<< |

# Examples of Bitwise operators

a = 10

b = 4

# Print bitwise AND operation

print (a & b)

# Print bitwise OR operation

print (a | b)

# Print bitwise NOT operation

print (~a)

# print bitwise XOR operation

print (a ^ b)

# print bitwise right shift operation

print(a >> 2)

# print bitwise left shift operation

Print (a << 2)

**5. Assignment operators:**

Assignment operators are used to assign values to the variables.

|  |  |  |
| --- | --- | --- |
| **OPERATOR** | **DESCRIPTION** | **SYNTAX** |
| = | Assign value of right side of expression to left side operand | x = y + z |
| += | Add AND: Add right side operand with left side operand and then assign to left operand | a+=b     a=a+b |
| -= | Subtract AND: Subtract right operand from left operand and then assign to left operand | a-=b       a=a-b |
| \*= | Multiply AND: Multiply right operand with left operand and then assign to left operand | a\*=b       a=a\*b |
| /= | Divide AND: Divide left operand with right operand and then assign to left operand | a/=b         a=a/b |
| %= | Modulus AND: Takes modulus using left and right operands and assign result to left operand | a%=b   a=a%b |
| //= | Divide(floor) AND: Divide left operand with right operand and then assign the value(floor) to left operand | a//=b       a=a//b |
| \*\*= | Exponent AND: Calculate exponent(raise power) value using operands and assign value to left operand | a\*\*=b     a=a\*\*b |
| &= | Performs Bitwise AND on operands and assign value to left operand | a&=b     a=a&b |
| |= | Performs Bitwise OR on operands and assign value to left operand | a|=b         a=a|b |
| ^= | Performs Bitwise xOR on operands and assign value to left operand | a^=b       a=a^b |
| >>= | Performs Bitwise right shift on operands and assign value to left operand | a>>=b     a=a>>b |
| <<= | Performs Bitwise left shift on operands and assign value to left operand | a= a << b |

**6. Special operators:**

There are some special type of operators like-

Identity operators-  
**is** and **is not** are the identity operators both are used to check if two values are located on the same part of the memory. Two variables that are equal does not imply that they are identical.

**is** True if the operands are identical

**is not** True if the operands are not identical

# Examples of Identity operators

a1 = 3

b1 = 3

a2 = 'Geeksforgeeks'

b2 = 'Geeksforgeeks'

a3 = [1, 2, 3]

b3 = [1, 2, 3]

print (a1 is not b1)

print (a2 is b2)

# Output is False, since lists are mutable.

print (a3 is b3)

Membership operators-  
**in** and **not in** are the membership operators; used to test whether a value or variable is in a sequence.

**in** True if value is found in the sequence

**not in** True if value is not found in the sequence

**Operands:**

In Python, operators are special symbols that designate that some sort of computation should be performed. The values that an operator acts on are called operands. A sequence of operands and operators, like a + b - 5, is called an expression.

**Order of operations:**

The following table lists all operators from highest precedence to lowest

|  |  |
| --- | --- |
| operator precedence rule in Python | |
| Operators | Meaning |
| () | Parentheses |
| \*\* | Exponent |
| +x, -x, ~x | Unary plus, Unary minus, Bitwise NOT |
| \*, /, //, % | Multiplication, Division, Floor division, Modulus |
| +, - | Addition, Subtraction |
| <<, >> | Bitwise shift operators |
| & | Bitwise AND |
| ^ | Bitwise XOR |
| | | Bitwise OR |
| ==, !=, >, >=, <, <=, is, is not, in, not in | Comparisons, Identity, Membership operators |
| not | Logical NOT |
| and | Logical AND |
| or | Logical OR |

Python Conditional Statements

**How to use “if condition” in conditional structures**

Decision structures evaluate multiple expressions which produce TRUE or FALSE as outcome. You need to determine which action to take and which statements to execute if outcome is TRUE or FALSE otherwise.

Following is the general form of a typical decision making structure found in most of the programming languages −



Python programming language assumes any non-zero and non-null values as TRUE, and if it is either zero or null, then it is assumed as FALSE value.

**if statement (One-Way Decisions)**

An if statement consists of a boolean expression followed by one or more statements.

Ex:

if(i<100):

print (“Hello world”)

**if.. else statement (Two-way Decisions)**

An if statement can be followed by an optional else statement, which executes when the boolean expression is FALSE.

if (i<100):

print (“Hello world”)

else:

print (“This is else case”)

**if.. elif... else statement (Multi-way)**

An elif statement is used to check the condition if another condition is failed.

If(i<100):

print(“Hello world”)

elif(i<40):

print(“This is new world”)

else:

print(“This is another world”)

**Nested IF Statement**

There may be a situation when you want to check for another condition after a condition resolves to true. In such a situation, you can use the nested ifconstruct.

In a nested if construct, you can have an if...elif...else construct inside another if...elif...else construct

The syntax of the nested if...elif...else construct may be –

if expression1:

statement(s)

if expression2:

statement(s)

elif expression3:

statement(s)

elif expression4:

statement(s)

else:

statement(s)

else:

statement(s)

Python LOOPS

**How to use “While Loop” and  “For Loop”**

Python programming language provides following types of loops to handle looping requirements. Python provides three ways for executing the loops. While all the ways provide similar basic functionality, they differ in their syntax and condition checking time.

1. While Loop:

In python, while loop is used to execute a block of statements repeatedly until a given a condition is satisfied. And when the condition becomes false, the line immediately after the loop in program is executed.

Syntax:

While expression:

Statement(s)

All the statements indented by the same number of character spaces after a programming construct are considered to be part of a single block of code. Python uses indentation as its method of grouping statements.

# Python program to illustrate

# while loop

count = 0

While (count < 3):

count = count + 1

print ("Hello Geek")

**Using else statement with while loops:**

As discussed above, while loop executes the block until a condition is satisfied. When the condition becomes false, the statement immediately after the loop is executed.  
The else clause is only executed when your while condition becomes false. If you break out of the loop, or if an exception is raised, it won’t be executed.

while condition:

# execute these statements

else:

# execute these statements

**Single statement while block:**

Just like the if block, if the while block consists of a single statement then we can declare the entire loop in a single line as shown below

# Python program to illustrate

# Single statement while block

Ex:

Count = 0

While (Count == 0): print ("Hello Geek")

**2. for in Loop:**

For loops are used for sequential traversal. For example: traversing a list or string or array etc. In Python, there is no C style for loop, i.e., for (i=0; i<n; i++). There is “for in” loop which is similar to [for each](https://www.geeksforgeeks.org/g-fact-40-foreach-in-c-and-java/) loop in other languages. Let us learn how to use for in loop for sequential traversals.

Syntax:

for i in range(start, stop):

Statements

**Iterating by index of sequences:**

We can also use the index of elements in the sequence to iterate. The key idea is to first calculate the length of the list and in iterate over the sequence within the range of this length.

# Python program to illustrate

# iterating by index

list = ["geeks", "for", "geeks"]

for index in range (len (list)):

print list [index]

**Using else statement with for loops:**

We can also combine else statement with for loop like in while loop. But as there is no condition in for loop based on which the execution will terminate so the else block will be executed immediately after for block finishes execution.

# Python program to illustrate

# combining else with for

list = ["geeks", "for", "geeks"]

for index in range (len (list)):

print list [index]

else:

print "Inside Else Block"

**Nested Loops:**

Python programming language allows to use one loop inside another loop.

Syntax:

for iterator\_var in sequence:

for iterator\_var in sequence:

statements(s)

statements(s)

while expression:

while expression:

statement(s)

statement(s)

A final note on loop nesting is that we can put any type of loop inside of any other type of loop. For example a for loop can be inside a while loop or vice versa.

# Python program to illustrate

# nested for loops in Python

from \_\_future\_\_ import print function

for i in range (1, 5):

for j in range (i):

print (i, end=' ')

print ()

**Loop Control Statements:**

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed. Python supports the following control statements.

**Continue Statement:**

It returns the control to the beginning of the loop.

# Prints all letters except 'e' and’s’

for letter in 'geeksforgeeks':

if letter == 'e' or letter == 's':

continue

print 'Current Letter :', letter

var = 10

Output:

Current Letter : g

Current Letter : k

Current Letter : f

Current Letter : o

Current Letter : r

Current Letter : g

Current Letter : k

**Break Statement:**

It brings control out of the loop.

for letter in 'geeksforgeeks':

# break the loop as soon it sees 'e'

# or’s’

if letter == 'e' or letter ==‘s’:

break

print 'Current Letter:’ letter

Output:

Current Letter: e

**Pass Statement:**

We use pass statement to write empty loops. Pass is also used for empty control statement, function and classes.

# An empty loop

for letter in 'geeksforgeeks':

pass

print 'Last Letter:’ letter

Output:

Last Letter: s

**How to use for loop to repeat the same statement over and again**

from itertools import count

for i in count ():

# ...

pass

The above code snippet shows the infinite loop.

Learning Python Strings

**Accessing Characters in Strings**

We can access individual characters using indexing and a range of characters using slicing. Index starts from 0. Trying to access a character out of index range will raise an Index Error. The index must be an integer. We can't use float or other types, this will result into Type Error.

Python allows negative indexing for its sequences.

The index of -1 refers to the last item, -2 to the second last item and so on. We can access a range of items in a string by using the slicing operator (colon).

str = 'programiz'

print ('str = ', str)

#first character

print (‘str [0] = ', str[0])

#last character

print ('str[-1] = ', str[-1])

#slicing 2nd to 5th character

print ('str[1:5] = ', str[1:5])

#slicing 6th to 2nd last character

print ('str[5:-2] = ', str[5:-2])

Output:

str = programiz

str [0] = p

str [-1] = z

str [1:5] = rogr

str [5:-2] = am

**Various String Operators**

|  |  |  |
| --- | --- | --- |
| Operator | Description | Operation |
| + | Concatenates (joins) *string1* and *string2* | string1 + string2 |
| \* | Repeats the string for as many times as specified by *x* | string \* x |
| [] | Slice — Returns the character from the index provided at *x*. | string[x] |
| [:] | Range Slice — Returns the characters from the range provided at *x:y*. | string[x:y] |
| in | Membership — Returns True if *x* exists in the string. Can be multiple characters. | x in string |
| not in | Membership — Returns True if *x* does *not* exist in the string. Can be multiple characters. | x not in string |
| r | Suppresses an escape sequence (\*x*) so that it is actually rendered. In other words, it prevents the escape character from being an escape character. | r"\x" |
| % | Performs string formatting. It can be used as a placeholder for another value to be inserted into the string. The %symbol is a prefix to another character (*x*) which defines the type of value to be inserted. The value to be inserted is listed at the end of the string after another % character.   |  |  | | --- | --- | | Character | Description | | %c | Character. | | %s | String conversion via str () prior to formatting. | | %i | Signed decimal integer. | | %d | Signed decimal integer. | | %u | Unsigned decimal integer. | | %o | Octal integer. | | %x | Hexadecimal integer using lowercase letters. | | %X | Hexadecimal integer using uppercase letters. | | %e | Exponential notation with lowercase e. | | %E | Exponential notation with uppercase e. | | %f | Floating point real number. | | %g | The shorter of %f and %e. | | %G | The shorter of %f and %E. | |  |

**Some more examples**

1. a = "Tea” + "Leaf"

print (a)

Output:

Tea Leaf

2. a = "Bee” \* 3

print (a)

Output:

Bee Bee Bee

3. a = "Sea"

print (a[1])

Output:

e

4. a = "Mushroom"

print (a [4:8])

Output:

Room

5. a = "Mushroom"

print ("m" in a)

print ("b" in a)

print ("shroo" in a)

Output:

True

False

True

6. a = "Mushroom"

print ("m" not in a)

print ("b" not in a)

print ("shroo" not in a)

Output:

False

True

False

7. a = "1" + "\t" + "Bee"

b = "2" + r"\t" + "Tea"

print (a)

print (b)

Output:

1 Bee

2\tTea

8. a = "Hi %s" % ("Homer")

print(a)

Output:

Hi Homer

**Python String replace () Method**

The replace () method can take maximum of 3 parameters:

* old - old substring you want to replace
* new - new substring which would replace the old substring
* count (optional) - the number of times you want to replace the old substring with the new substring

If count is not specified, replace () method replaces all occurrences of the old substring with the new substring.

## Return Value from replace ()

The replace () method returns a copy of the string where old substring is replaced with the new substring. The original string is unchanged.

If the old substring is not found, it returns the copy of the original string.

Ex:

song = 'cold, cold heart'

replaced\_song = song.replace ('o', 'e')

print ('Replaced string:’ replaced\_song)

Output:

Replaced string:celd ,celd heart

**Changing upper and lower case strings**

1. lower ():

The string lower () method converts all uppercase characters in a string into lowercase characters and returns it**.**

The syntax of lower () method is:

String. lower ()

The lower () method doesn't take any parameters.

The lower () method returns the lowercased string from the given string. It converts all uppercase characters to lowercase.

If no uppercase characters exist, it returns the original string

Ex:

String=”THIS IS VIDYANATH”

Changed\_string=String.lower ()

print (Changed\_string)

output:

“this is vidyanath”

2. upper ():

The string upper () method converts all lowercase characters in a string into uppercase characters and returns it.

The syntax of upper () method is:

String. upper ()

The upper () method returns the uppercased string from the given string. It converts all lowecase characters to uppercase.

If no lowercase characters exist, it returns the original string.

Ex:

String = "this should be uppercase!"

print (String.upper ())

output:

“THIS SHOULD BE UPPERCASE”

**Using “join” function for the string**

The join () is a string method which returns a string concatenated with the elements of an iterable.

The join () method provides a flexible way to concatenate string. It concatenates each element of an iterable (such as list, string and tuple) to the string and returns the concatenated string.

The syntax of join () is:

String. Join (iterable)

The join () method takes an iterable - objects capable of returning its members one at a time

Some of the example of iterables are:

* **Native datatypes** - [List](https://www.programiz.com/python-programming/list), [Tuple](https://www.programiz.com/python-programming/tuple), [String](https://www.programiz.com/python-programming/string), [Dictionary](https://www.programiz.com/python-programming/dictionary) and [Set](https://www.programiz.com/python-programming/set)
* The join () method returns a string concatenated with the elements of an iterable.
* If the iterable contains any non-string values, it raises a Type Error exception.

Ex:

NumList= [1, 2, 3, 4]

Separator=’-’

print (Seperator.join (NumList))

output:

1-2-3-4

**Reversing String**

There is no built in reverse function for Python's str object. We can reverse a string by doing [begin: end: step] - by leaving begin and end off and specifying a step of -1, it reverses a string.

Ex:

String=String [::-1]

**Split Strings**

**split ()** method returns a list of strings after breaking the given string by the specified separator.

Str.split (separator, maxsplit)

separator:

The is a delimiter. The string splits at this specified separator. If is not provided then any white space is a separator.

maxsplit:

It is a number, which tells us to split the string into maximum of provided number of times. If it is not provided then there is no limit.

Ex:

text = 'geeks for geeks'

# Splits at space

print (text.split ())

Output:

[‘geeks’,’for’,’geeks’]

Sequence or Collections in PYTHON

**Sequence:**

In [Python](https://artofproblemsolving.com/wiki/index.php/Python), sequence is the generic term for an ordered set.  There are several types of sequences in Python, the following three are the most important.

[Lists](https://artofproblemsolving.com/wiki/index.php/List) are the most versatile sequence type. The elements of a list can be any object, and lists are mutable - they can be changed. Elements can be reassigned or removed, and new elements can be inserted.

[Tuples](https://artofproblemsolving.com/wiki/index.php/Tuple) are like lists, but they are immutable - they can't be changed.

[Strings](https://artofproblemsolving.com/wiki/index.php/String) are a special type of sequence that can only store [characters](https://artofproblemsolving.com/wiki/index.php?title=Character&action=edit&redlink=1), and they have a special notation. However, all of the sequence operations described below can also be used on strings.

**Unicode Strings:**

**Unicode** is a standard for working with a wide range of characters.

For Python 2, strings that contain Unicode characters must start with ‘u’ in front of the string.

Ex: aa=u”I U”

The ‘u’ makes the string a Unicode datatype. Without the ‘u’, string is just byte sequence.

Sometimes when you print Unicode strings, you may get a error like this:

UnicodeEncodeError: 'ascii' codec can't encode character u'\u03b1' in position 16: ordinal not in range (128).

**range () and xrange ():**

range () returns – the **list** as return type.  
xrange () returns – **xrange ()** object.

Python Lists

**Lists are mutable**

Lists are mutable types. We can change them.Custom classes are generally mutable.

We can declare any list by using “square brackets ([])”.Mutable objects are easy to change.

Ex:

List= [1, 2, 3, 4]

**List indices**

The index () method searches an element in the list and returns its index.

In simple terms, index () method finds the given element in a [list](https://www.programiz.com/python-programming/list) and returns its position.

However, if the same element is present more than once, index () method returns its smallest/first position.

**Note:** Index in Python starts from 0 not 1.

The syntax of index () method for [list](https://www.programiz.com/python-programming/list) is:

list.index(element)

index () Parameters

The index method takes a single argument:

* **element** - element that is to be searched.
* The index () method returns the index of the element in the list.
* If not found, it raises a ValueError exception indicating the element is not in the list.

# vowels list

vowels = ['a', 'e', 'i', 'o', 'i', 'u']

# element 'e' is searched

index = vowels.index ('e')

# index of 'e' is printed

print ('The index of e:’ index)

# element 'i' is searched

index = vowels.index ('i')

# only the first index of the element is printed

print ('The index of i:', index)

**Traversing a list**

We can traverse a list. We can access a range of items in a list by using the slicing operator (colon).

my\_list = ['p','r','o','g','r','a','m','i','z']

# elements 3rd to 5th

print (my\_list [2:5])

# elements beginning to 4th

print (my\_list [:-5])

# elements 6th to end

print (my\_list [5 :])

# Elements beginning to end

print (my\_list [:])

Ex:

['o', 'g', 'r']

['p', 'r', 'o', 'g']

['a', 'm', 'i', 'z']

['p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z']

**List operations**

We can change or add elements in a list.We can delete or remove an element from a list.

We can test if an item exists in a list or not, using the keyword ‘in’.

Using a for loop we can iterate though each item in a list.

### List Membership Test

We can test if an item exists in a list or not, using the keyword ‘in’.

Ex:

my\_list = ['p','r','o','b','l','e','m']

# Output: True

print ('p' in my\_list)

# Output: False

print ('a' in my\_list)

# Output: True

print ('c' not in my\_list)

### Iterating Through a List

Using a for loop we can iterate though each item in a list.

Ex:

for fruit in ['apple','banana','mango']:

print ("I like", fruit)

**Slices**

my\_list = ['p','r','o','g','r','a','m','i','z']

# elements 3rd to 5th

print (my\_list [2:5])

# elements beginning to 4th

print (my\_list [:-5])

# elements 6th to end

print (my\_list [5 :])

# Elements beginning to end

print (my\_list [:])

**Methods**

* append ():

Used for appending and adding elements to List.It is used to add elements to the last position of List.

List.append (element)

Ex:

# Adds List Element as value of List.

List = ['Mathematics', 'chemistry', 1997, 2000]

List.append(20544)

print (List)

o/p:

['Mathematics', 'chemistry', 1997, 2000, 20544]

* **insert():**

Inserts an elements at specified position.

List.insert (position, element)

Ex:

List = ['Mathematics', 'chemistry', 1997, 2000]

# Insert at index 2 value 10087

List.insert (2, 10087)

print(List)

o/p:

['Mathematics', 'chemistry', 10087, 1997, 2000]

* **extend():**

Adds contents of List2 to the end of List1.

List1.extend (List2)

Ex:

List1= [1, 2, 3]

List2= [1, 2, 3, 4]

List1.extend (List2)

Print (List1)

Output:

[1, 2, 3, 1, 2, 3, 4]

* **sum() :**

Calculates sum of all the elements of List

Ex:

List1= [1, 2, 3, 4]

print (sum (List1))

Output:

10

* **count():**

Calculates total occurrence of given element of List.

List. count (element)

Ex:

List = [1, 2, 3, 1, 2, 1, 2, 3, 2, 1]

print (List.count(1))

Output:

4

* **length:**

Calculates total length of List.

len (List)

* **index():**

Returns the index of first occurrence. Start and End index are not necessary parameters.

Index=List.index (element)

Ex:

List = [1, 2, 3, 1, 2, 1, 2, 3, 2, 1]

print (List.index (2))

* **min() :**

Calculates minimum of all the elements of List.

min (List)

Ex:

List = [2.3, 4.445, 3, 5.33, 1.054, 2.5]

print (min (List))

Output:

1.054

* **max():**

Calculates maximum of all the elements of List.

max (List)

Ex:

List = [2.3, 4.445, 3, 5.33, 1.054, 2.5]

print (max (List))

Output:

5.33

* **reverse():**

Sort the given data structure (both tuple and list) in ascending order. Key and reverse\_flag are not necessary parameter and reverse\_flag is set to False, if nothing is passed through sorted ().

#Reverse flag is set True

List.sort(reverse=True)

#List.sort ().reverse (), reverses the sorted list

print (List)

Ex:

List = [2.3, 4.445, 3, 5.33, 1.054, 2.5]

print (List.sort ().reverse ())

print (List)

Output:

[5.33, 4.445, 3, 2.5, 2.3, 1.054]

* **del() :**

Element to be deleted is mentioned using list name and index.

#Element to be deleted

del list [index]

Ex:

List = [2.3, 4.445, 3, 5.33, 1.054, 2.5]

del List[0]

print (List)

Output:

[4.445, 3, 5.33, 1.054, 2.5]

* **remove():**

Element to be deleted is mentioned using list name and element.

List.remove(element)

Ex:

List = [2.3, 4.445, 3, 5.33, 1.054, 2.5]

List.remove (3)

print (List)

* **Map**

**map()** function returns a list of the results after applying the given function to each item of a given iterable (list, tuple etc.)

map (fun, iter)

**fun:** It is a function to which map passes each element of given iterable.  
**iter:** It is a iterable which is to be mapped.

**NOTE:** You can pass one or more iterable to the map () function.

**NOTE:**The returned value from map () (map object) then can be passed to functions like list () (to create a list), set () (to create a set).

Ex:

# Python program to demonstrate working of map.

# Return double of n

def addition (n):

return n + n

# We double all numbers using map ()

numbers = (1, 2, 3, 4)

result = map (addition, numbers)

print (list (result))

Output:

[2, 4, 6, 8]

* **filter**

The filter () method filters the given sequence with the help of a function that tests each element in the sequence to be true or not.

filter (function, sequence)

Parameters:

function:

function that tests if each element of sequence is True or Not.

sequence:

sequence which needs to be filtered, it can be sets, tuples, lists or containers of any Iterators.

Returns:

returns an iterator that is already filtered.

# function that filters vowels

def fun (variable):

letters = ['a', 'e', 'i', 'o', 'u']

if (variable in letters):

return True

else:

return False

# sequence

sequence = ['g', 'e', 'e', 'j', 'k', 's', 'p', 'r']

# using filter function

filtered = filter(fun, sequence)

print ('The filtered letters are:')

for s in filtered:

print(s)

Output:

The filtered letters are

e

e

Python TUPLE

* **Advantages of Tuple over List**
  + Tuples operation has smaller size than that of list, which makes it a bit faster but not that much to mention about until you have a huge number of elements
  + It makes your code safer if you “write-protect” data that does not need to be changed. Using a tuple instead of a list is like having an implied assert statement that this data is constant, and that special thought (and a specific function) is required to override that.
  + Some tuples can be used as dictionary keys (specifically, tuples that contain immutable values like strings, numbers, and other tuples). Lists can never be used as dictionary keys, because lists are not immutable.
  + A tuple is **immutable** whereas a list is **mutable.**
* **Packing and Unpacking**

We use two operators \*(tuples) and \*\*(dictionaries) for Packing and Unpacking.

**Packing**  
When we don’t know how many arguments need to be passed to a python function, we can use packing to pack all arguments in a tuple.

**Ex:**

# A Python program to demonstrate use of packing

#This function uses packing to sum unknown number of arguments

def mySum(\*args):

sum = 0

for i in range(0, len(args)):

sum = sum + args[i]

return sum

# Driver code

print(mySum(1, 2, 3, 4, 5))

print(mySum(10, 20))