MCA III Semester

Machine Learning Lab Manual

****

Introduction to Machine Learning

To solve a problem on a computer we need an algorithm. For the same task, there may be various algorithms and we may be interested in finding the most efficient one, requiring the least number of instructions or memory or both.

For some tasks, however, we do not have an algorithm – for example, to tell spam emails from legitimate mails. We know what the input is: to distinguish spam emails from legitimate emails. We know the input: an email document. We know what the output should be: a yes/no output indicating whether the message is spam or not. We do not know how to transform the input to output. What can be considered spam changes in time and from individual to individual.

What we lack in knowledge, we make up for in data. We would expect the computer (machine) to extract automatically the algorithm for this task. There are many applications for which we do not have an algorithm but do have example data.

Machine Learning is programming computers to optimize a performance criterion using example data or past experience. We have a model defined up to some parameters, and learning is the execution of a computer program to optimize the parameters of the model using training data or past experience. The model may be predictive to make predictions in the future or descriptive to gain knowledge from data, or both.

Machine Learning uses the theory of statistics in building mathematical models, because the core task is making inference from a sample.

Chapter-01: Introduction to Python

Python is an open source, high-level programming language developed by Guido van Rossum in the late 1980s and presently administered by Python Software Foundation. It came from the ABC language that he helped create early on in his career.

Python is a powerful language that you can use to create games, write GUIs, and develop web applications. It is a high-level language. Reading and writing codes in Python is much like reading and writing regular English statements. Because they are not written in machine-readable language, Python programs need to be processed before machines can run them.

Python is an interpreted language.

This means that every time a program is run, its interpreter runs through the code and translates it into machine-readable byte code.

Python is an object-oriented language that allows users to manage and control data structures or objects to create and run programs.

Everything in Python is, in fact, first class. All objects, data types, functions, methods, and classes take equal position in Python.

Programming languages are created to satisfy the needs of programmers and users for an effective tool to develop applications that impact lives, lifestyles, economy, and society. They help make lives better by increasing productivity, enhancing communication, and improving efficiency. Languages die and become obsolete when they fail to live up to expectations and are replaced and superseded by languages that are more powerful.

Python is a programming language that has stood the test of time and has remained relevant across industries and businesses and among programmers, and individual users. It is a living, thriving, and highly useful language that is highly recommended as a first programming language for those who want to dive into and experience programming.

Advantages of Using Python

Here are reasons why you would prefer to learn and use Python over other high level languages:

1) **Readability:** Python programs use clear, simple, and concise instructions that are easy to read even by those who have no substantial programming background. Programs written in Python are, therefore, easier to maintain, debug, or enhance.

2) **Higher productivity:** Codes used in Python are considerably shorter, simpler, and less verbose than other high level programming languages such as Java and C++. In addition, it has well-designed built-in features and standard library as well as access to third party modules and source libraries. These features make programming in Python more efficient.

3) **Less learning time:** Python is relatively easy to learn. Many find Python a good first language for learning programming because it uses simple syntax and shorter codes.

4) **Runs across different platforms:** Python works on Windows, Linux/UNIX, Mac OS X, other operating systems and smallform devices. It also runs on microcontrollers used in appliances, toys, remote controls, embedded devices, and other similar devices.

Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language. It was created by Guido van Rossum during 1985- 1990. Like Perl, Python source code is also available under the GNU General Public License (GPL). This **tutorial** gives enough understanding on **Python programming** language.

Why to Learn Python?

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

Python is a MUST for students and working professionals to become a great Software Engineer specially when they are working in Web Development Domain. I will list down some of the key advantages of learning Python:

- **Python is Interpreted** − Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
- **Python is Interactive** − You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
- **Python is Object-Oriented** − Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
- **Python is a Beginner's Language** − Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

Characteristics of Python

Following are important characteristics of **Python Programming** −

- It supports functional and structured programming methods as well as OOP.
- It can be used as a scripting language or can be compiled to byte-code for building large applications.
- It provides very high-level dynamic data types and supports dynamic type checking.
- It supports automatic garbage collection.
- It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

Hello World using Python.

Just to give you a little excitement about Python, I'm going to give you a small conventional Python Hello World program, You can try it using Demo link.

```
print ("Hello, Python!");
```

Applications of Python

As mentioned before, Python is one of the most widely used language over the web. I'm going to list few of them here:

- **Easy-to-learn** − Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.
- **Easy-to-read** − Python code is more clearly defined and visible to the eyes.
- **Easy-to-maintain** − Python's source code is fairly easy-to-maintain.
- **A broad standard library** − Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
- **Interactive Mode** − Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
- **Portable** − Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- **Extendable** − You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- **Databases** − Python provides interfaces to all major commercial databases.
- **GUI Programming** − Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.
- **Scalable** − Python provides a better structure and support for large programs than shell scripting.

Prerequisites

You should have a basic understanding of Computer Programming terminologies. A basic understanding of any of the programming languages is a plus.

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

- **Python is Interpreted** − Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
- **Python is Interactive** − You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
- **Python is Object-Oriented** − Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
- **Python is a Beginner's Language** − Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

History of Python

Python was developed by Guido van Rossum in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the Netherlands.

Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, SmallTalk, and Unix shell and other scripting languages.

Python is copyrighted. Like Perl, Python source code is now available under the GNU General Public License (GPL).

Python is now maintained by a core development team at the institute, although Guido van Rossum still holds a vital role in directing its progress.

Python Features

Python's features include −

- **Easy-to-learn** − Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.
- **Easy-to-read** − Python code is more clearly defined and visible to the eyes.
- **Easy-to-maintain** − Python's source code is fairly easy-to-maintain.
- **A broad standard library** − Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
- **Interactive Mode** − Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
- **Portable** − Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- **Extendable** − You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- **Databases** − Python provides interfaces to all major commercial databases.
- **GUI Programming** − Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.
- **Scalable** − Python provides a better structure and support for large programs than shell scripting.

Apart from the above-mentioned features, Python has a big list of good features, few are listed below −

- It supports functional and structured programming methods as well as OOP.
- It can be used as a scripting language or can be compiled to byte-code for building large applications.
- It provides very high-level dynamic data types and supports dynamic type checking.
- It supports automatic garbage collection.
- It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

Chapter02: Installing Python

Python is available on a wide variety of platforms including Linux and Mac OS X. Let's understand how to set up our Python environment.

Local Environment Setup

Open a terminal window and type "python" to find out if it is already installed and which version is installed.

- Unix (Solaris, Linux, FreeBSD, AIX, HP/UX, SunOS, IRIX, etc.)
- Win 9x/NT/2000
- Macintosh (Intel, PPC, 68K)
- OS/2
- DOS (multiple versions)
- PalmOS
- Nokia mobile phones
- Windows CE
- Acorn/RISC OS
- BeOS
- Amiga
- VMS/OpenVMS
- QNX
- VxWorks
- Psion
- Python has also been ported to the Java and .NET virtual machines

Getting Python

The most up-to-date and current source code, binaries, documentation, news, etc., is available on the official website of Python https://www.python.org/

You can download Python documentation from https://www.python.org/doc/ . The documentation is available in HTML, PDF, and PostScript formats.

Installing Python

Python distribution is available for a wide variety of platforms. You need to download only the binary code applicable for your platform and install Python.

If the binary code for your platform is not available, you need a C compiler to compile the source code manually. Compiling the source code offers more flexibility in terms of choice of features that you require in your installation.

Here is a quick overview of installing Python on various platforms −

Unix and Linux Installation

Here are the simple steps to install Python on Unix/Linux machine.

- Open a Web browser and go to https://www.python.org/downloads/ .
- Follow the link to download zipped source code available for Unix/Linux.
- Download and extract files.
- Editing the *Modules/Setup* file if you want to customize some options.
- run ./configure script
- make
- make install

This installs Python at standard location */usr/local/bin* and its libraries at */usr/local/lib/pythonXX* where XX is the version of Python.

Windows Installation

Here are the steps to install Python on Windows machine.

- Open a Web browser and go to https://www.python.org/downloads/ .
- Follow the link for the Windows installer *python-XYZ.msi* file where XYZ is the version you need to install.
- To use this installer *python-XYZ.msi*, the Windows system must support Microsoft Installer 2.0. Save the installer file to your local machine and then run it to find out if your machine supports MSI.
- Run the downloaded file. This brings up the Python install wizard, which is really easy to use. Just accept the default settings, wait until the install is finished, and you are done.

Setting up PATH

Programs and other executable files can be in many directories, so operating systems provide a search path that lists the directories that the OS searches for executables.

The path is stored in an environment variable, which is a named string maintained by the operating system. This variable contains information available to the command shell and other programs.

The **path** variable is named as PATH in Unix or Path in Windows (Unix is case sensitive; Windows is not).

In Mac OS, the installer handles the path details. To invoke the Python interpreter from any particular directory, you must add the Python directory to your path.

Setting path at Unix/Linux

To add the Python directory to the path for a particular session in Unix −

- **In the csh shell** − type setenv PATH "$PATH:/usr/local/bin/python" and press Enter.
- **In the bash shell (Linux)** − type export PATH="$PATH:/usr/local/bin/python" and press Enter.
- **In the sh or ksh shell** − type PATH="$PATH:/usr/local/bin/python" and press Enter.
- **Note** − /usr/local/bin/python is the path of the Python directory

Setting path at Windows

To add the Python directory to the path for a particular session in Windows −

At the command prompt − type path %path%;C:\Python and press Enter.

Note − C:\Python is the path of the Python directory

Python Environment Variables

Here are important environment variables, which can be recognized by Python −

Sr.No.	Variable & Description
1	**PYTHONPATH** It has a role similar to PATH. This variable tells the Python interpreter where to locate the module files imported into a program. It should include the Python source library directory and the directories containing Python source code. PYTHONPATH is sometimes preset by the Python installer.
2	**PYTHONSTARTUP** It contains the path of an initialization file containing Python source code. It is executed every time you start the interpreter. It is named as .pythonrc.py in Unix and it contains commands that load utilities or modify PYTHONPATH.
3	**PYTHONCASEOK** It is used in Windows to instruct Python to find the first case-insensitive match in an import statement. Set this variable to any value to activate it.
4	**PYTHONHOME** It is an alternative module search path. It is usually embedded in the PYTHONSTARTUP or PYTHONPATH directories to make switching module libraries easy.

Running Python

There are three different ways to start Python −

Interactive Interpreter

You can start Python from Unix, DOS, or any other system that provides you a command-line interpreter or shell window.

Enter **python** the command line.

Start coding right away in the interactive interpreter.

```
$python # Unix/Linux
or
python% # Unix/Linux
or
C:> python # Windows/DOS
```

Here is the list of all the available command line options −

Sr.No.	Option & Description
1	**-d** It provides debug output.
2	**-O** It generates optimized bytecode (resulting in .pyo files).
3	**-S** Do not run import site to look for Python paths on startup.
4	**-v** verbose output (detailed trace on import statements).
5	**-X** disable class-based built-in exceptions (just use strings); obsolete starting with version 1.6.
6	**-c cmd** run Python script sent in as cmd string
7	**file** run Python script from given file

Script from the Command-line

A Python script can be executed at command line by invoking the interpreter on your application, as in the following −

```
$python script.py # Unix/Linux

or

python% script.py # Unix/Linux

or

C: >python script.py # Windows/DOS
```

Note − Be sure the file permission mode allows execution.

Integrated Development Environment

You can run Python from a Graphical User Interface (GUI) environment as well, if you have a GUI application on your system that supports Python.

- **Unix** − IDLE is the very first Unix IDE for Python.
- **Windows** − PythonWin is the first Windows interface for Python and is an IDE with a GUI.
- **Macintosh** − The Macintosh version of Python along with the IDLE IDE is available from the main website, downloadable as either MacBinary or BinHex'd files.

If you are not able to set up the environment properly, then you can take help from your system admin. Make sure the Python environment is properly set up and working perfectly fine.

Note − All the examples given in subsequent chapters are executed with Python 2.4.3 version available on CentOS flavor of Linux.

We already have set up Python Programming environment online, so that you can execute all the available examples online at the same time when you are learning theory. Feel free to modify any example and execute it online.

Installing Python in Windows

To install Python, you must first download the installation package of your preferred version from this link: https://www.python.org/downloads/

On this page, you will be asked to choose between the two latest versions for Python 2 and 3: Python 3.5.1 and Python 2.7.11. Alternatively, if you are looking for a specific release, you can scroll down the page to find download links for earlier versions.

If you're using a Mac, you can download the installation package from this link: https://www.python.org/downloads/mac-osx/

Running the Installation file:

Once you're finished with the download, you can proceed to installation by clicking on the downloaded .exe file. Standard installation will include IDLE, pip, and documentation.

Python Download and Installation Instructions

You may want to print these instructions before proceeding, so that you can refer to them while downloading and installing Python. Or, just keep this document in your browser. You should read each step completely before performing the action that it describes.

This document shows downloading and installing Python 3.7.4 on Windows 10 in Summer 2019. **You should download and install the latest version of Python.** The current latest (as of Winter 2020) is Python 3.8.1.

Note: **You must install Java, Python, and Eclipse as all 64-bit applications**.

Python: Version 3.7.4

The Python download requires about 25 Mb of disk space; keep it on your machine, in case you need to re-install Python. When installed, Python requires about an additional 90 Mb of disk space.

Python: Version 3.7.4

The Python download requires about 25 Mb of disk space; keep it on your machine, in case you need to re-install Python. When installed, Python requires about an additional 90 Mb of disk space.

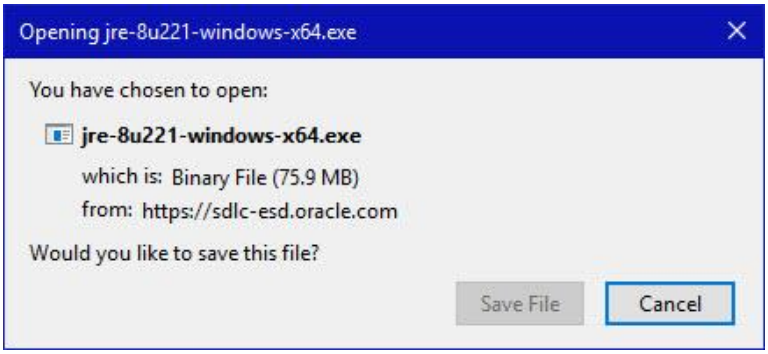
Downloading Python

1. Click Python Download.

   The following page will appear in your browser.

2. Click the **Windows** link (two lines below the **Download Python 3.7.4** button). The following page will appear in your browser.

Click on the **Download <u>Windows x86-64 executable installer</u>** link under the top-left **Stable Releases**. The following pop-up window titled **Opening python-3.74-amd64.exe** will appear.



1. Click the **Save File** button.
   The file named **python-3.7.4-amd64.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

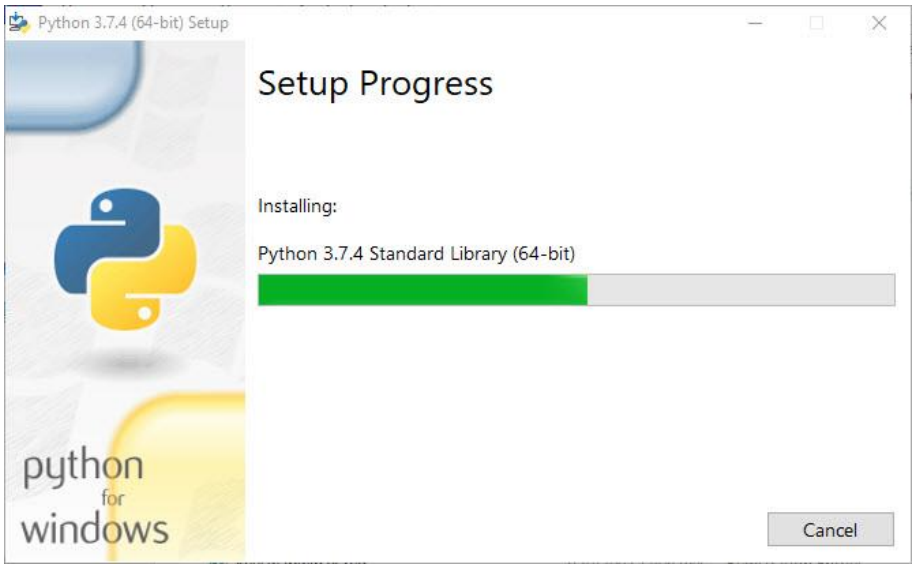
   

2. Move this file to a more permanent location, so that you can install Python (and reinstall it easily later, if necessary).
3. Feel free to explore this webpage further; if you want to just continue the installation, you can terminate the tab browsing this webpage.
4. Start the **Installing** instructions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

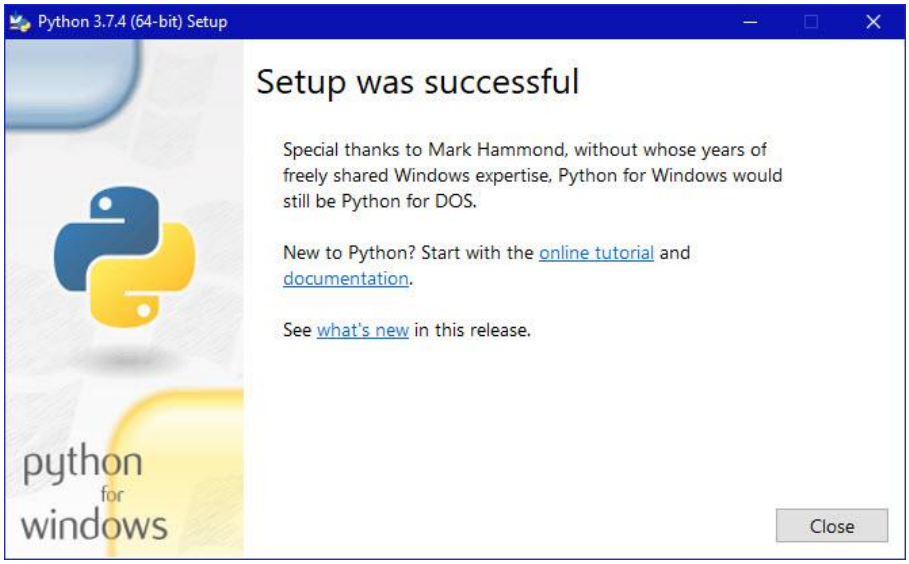
Installing

1. Double-click the icon labeling the file **python-3.7.4-amd64.exe**.
   A **Python 3.7.4 (64-bit) Setup** pop-up window will appear.

1. 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).
2. Highlight the **Install Now** (or **Upgrade Now**) message, and then click it.
   When run, a **User Account Control** pop-up window may appear on your screen. I could not capture its image, but it asks, **Do you want to allow this app to make changes to your device**.
3. Click the **Yes** button.
   A new **Python 3.7.4 (64-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 3.7.4 (64-bit) Setup** pop-up window will appear with a **Setup was successfuly** message.



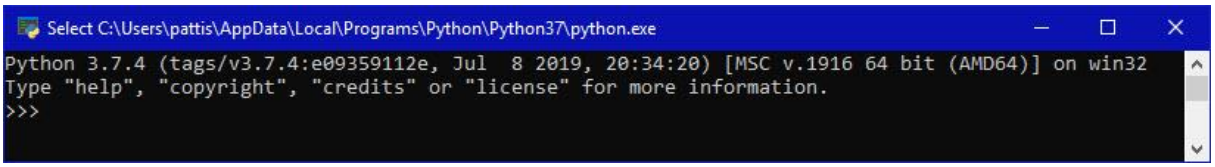
Click the **Close** button.

Python should now be installed.

Installation Verification

To try to verify installation,

1. Navigate to the directory
   C:\Users\Pattis\AppData\Local\Programs\Python\Python37 (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\python.exe** appears, and inside the window; on the first line is the text **Python 3.7.4 ...** (notice that it should also say 64 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-3.7.4.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).

https://docs.anaconda.com/anaconda/install/windows/

https://docs.anaconda.com/anaconda/user-guide/getting-started/

Anaconda Installation on Windows

1. Download the Anaconda installer.
2. RECOMMENDED: Verify data integrity with SHA-256. For more information on hashes, see What about cryptographic hash verification?
3. Double click the installer to launch.

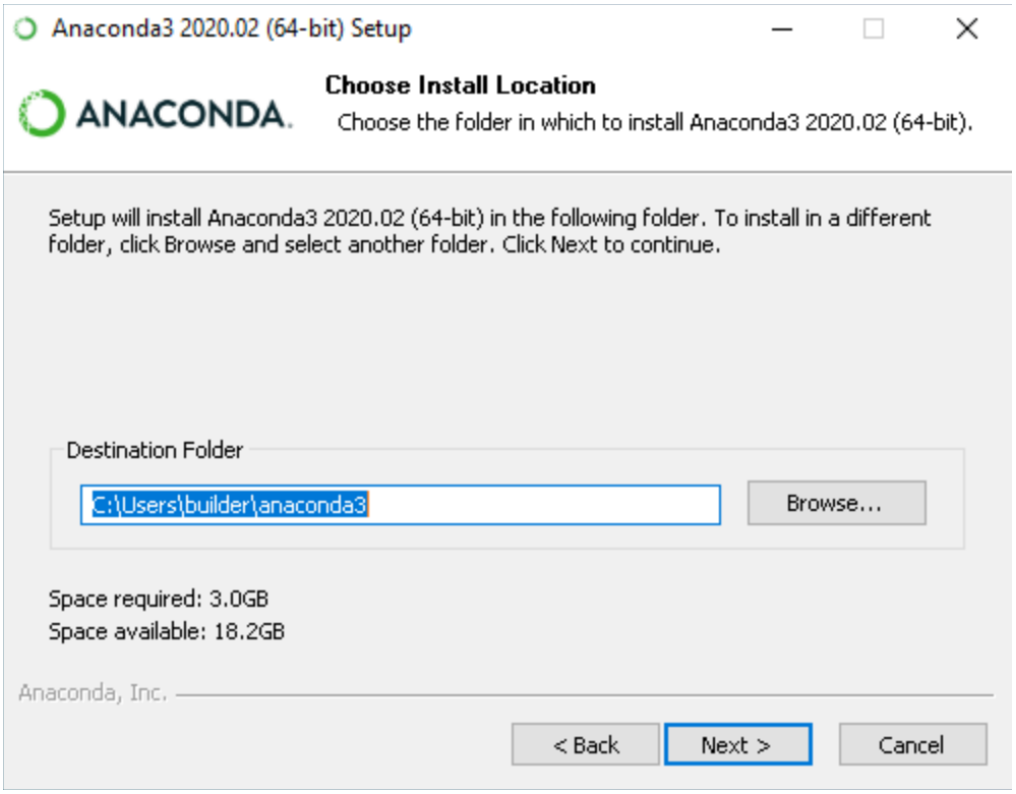
   Note: To prevent permission errors, do not launch the installer from the Favorites folder.

   Note: If you encounter issues during installation, temporarily disable your anti-virus software during install, then re-enable it after the installation concludes. If you installed for all users, uninstall Anaconda and re-install it for your user only and try again.

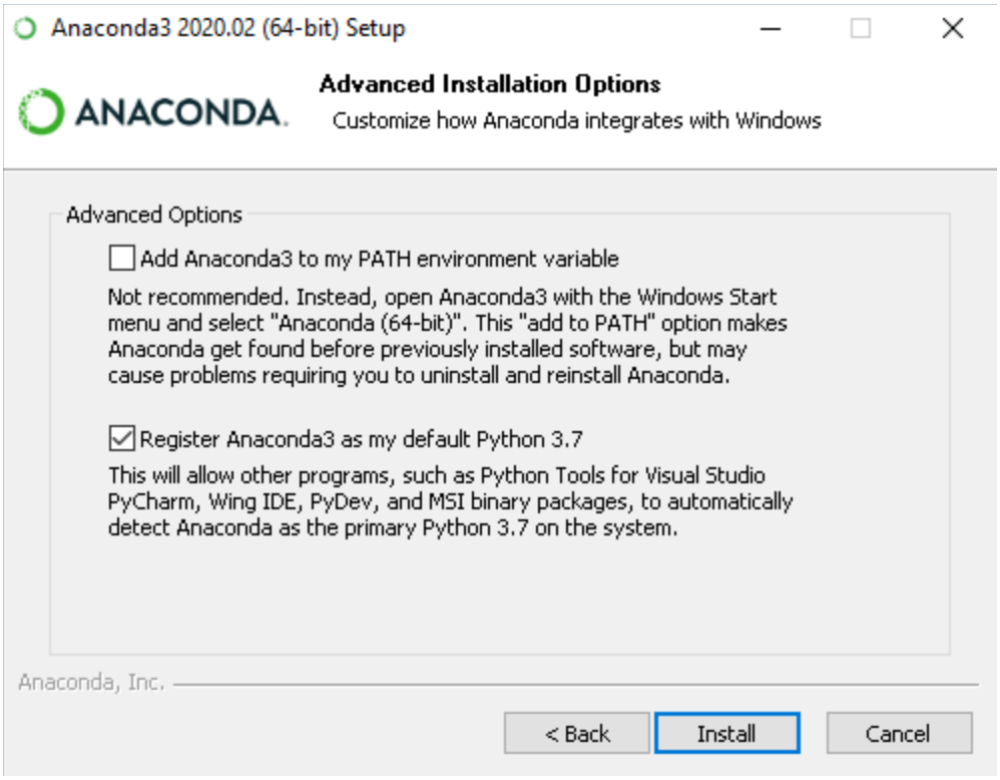
4. Click Next.
5. Read the licensing terms and click “I Agree”.
6. Select an install for “Just Me” unless you’re installing for all users (which requires Windows Administrator privileges) and click Next.
7. Select a destination folder to install Anaconda and click the Next button.

   Note: Install Anaconda to a directory path that does not contain spaces or unicode characters.

   Note: Do not install as Administrator unless admin privileges are required.



8. Choose whether to add Anaconda to your PATH environment variable. We recommend not adding Anaconda to the PATH environment variable, since this can interfere with other software. Instead, use Anaconda software by opening Anaconda Navigator or the Anaconda Prompt from the Start Menu.

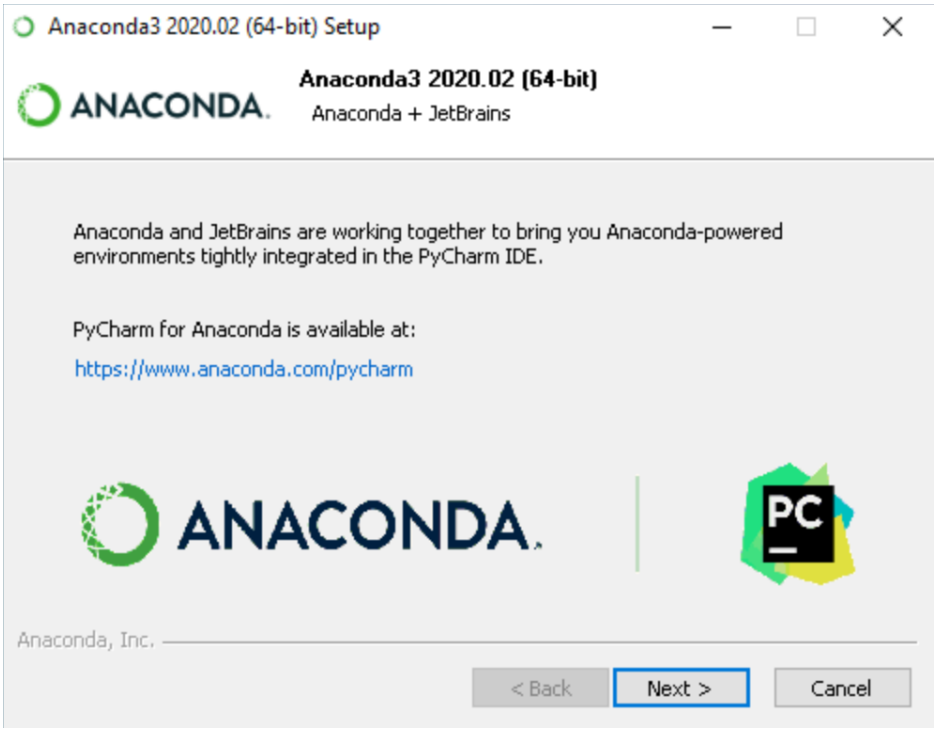


9. Choose whether to register Anaconda as your default Python. Unless you plan on installing and running multiple versions of Anaconda or multiple versions of Python, accept the default and leave this box checked.

10. Click the Install button. If you want to watch the packages Anaconda is installing, click Show Details.

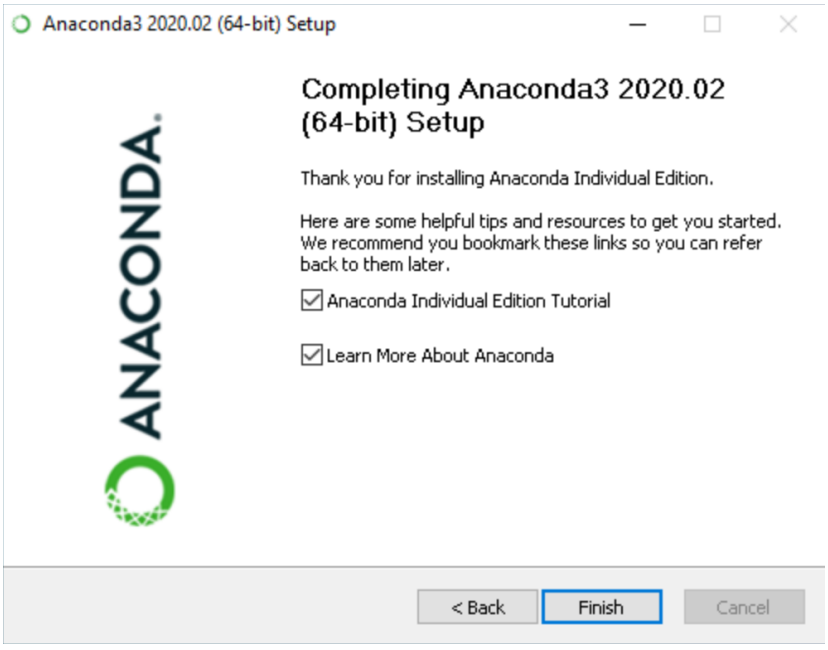
11. Click the Next button.

Optional: To install PyCharm for Anaconda, click on the link to https://www.anaconda.com/pycharm.



Or to install Anaconda without PyCharm, click the Next button.

13. After a successful installation you will see the “Thanks for installing Anaconda” dialog box:



14. If you wish to read more about Anaconda Cloud and how to get started with Anaconda, check the boxes “Learn more about Anaconda Cloud” and “Learn how to get started with Anaconda”. Click the Finish button.

15. **Verify your installation**.

Getting started with Anaconda

Anaconda Individual Edition contains conda and Anaconda Navigator, as well as Python and hundreds of scientific packages. When you installed Anaconda, you installed all these too.

Conda works on your command line interface such as Anaconda Prompt on Windows and terminal on macOS and Linux.

Navigator is a desktop graphical user interface that allows you to launch applications and easily manage conda packages, environments, and channels without using command-line commands.
You can try both conda and Navigator to see which is right for you to manage your packages and environments. You can even switch between them, and the work you do with one can be viewed in the other.

Try this simple programming exercise, with Navigator and the command line, to help you decide which approach is right for you.

When you’re done, see What’s next?.

Our first Python program: Hello, Anaconda!

1) Use Anaconda Navigator to launch an application. Then, create and run a simple Python program with Spyder and Jupyter Notebook.

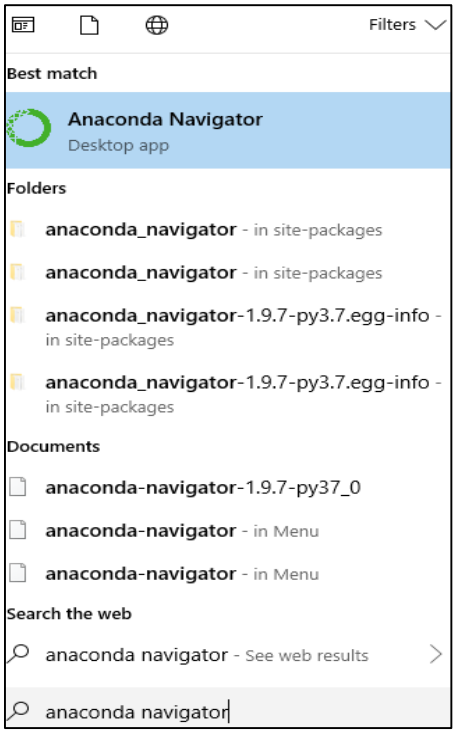
2) **Open Navigator**

Choose the instructions for your operating system.

- Windows.
- macOS.
- Linux.

Windows

From the Start menu, click the Anaconda Navigator desktop app.



macOS: Open Launchpad, then click the Anaconda Navigator icon.

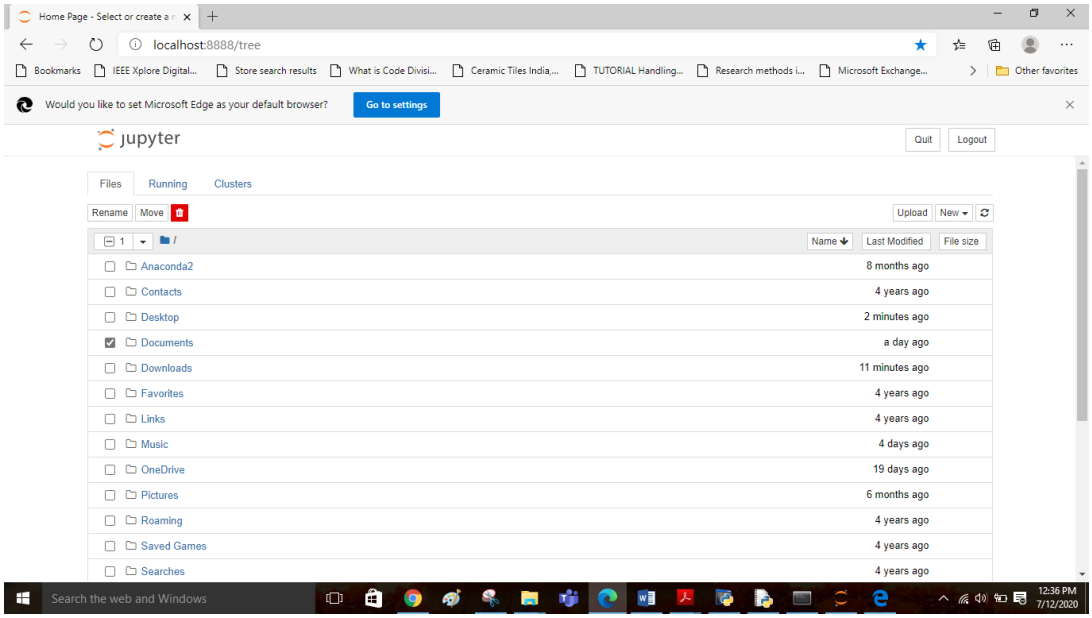
Linux: Open a terminal window and type anaconda-navigator.

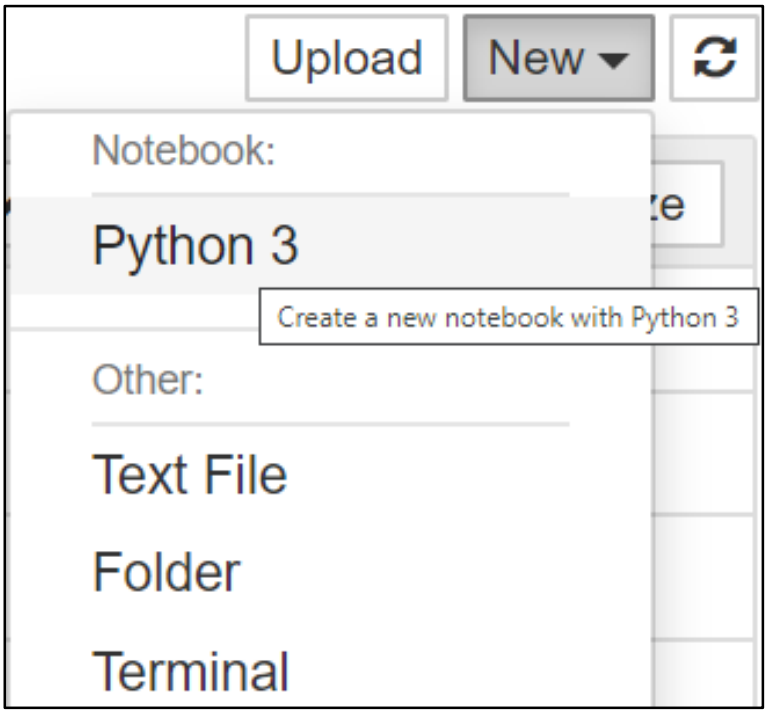
Run Python in a Jupyter Notebook

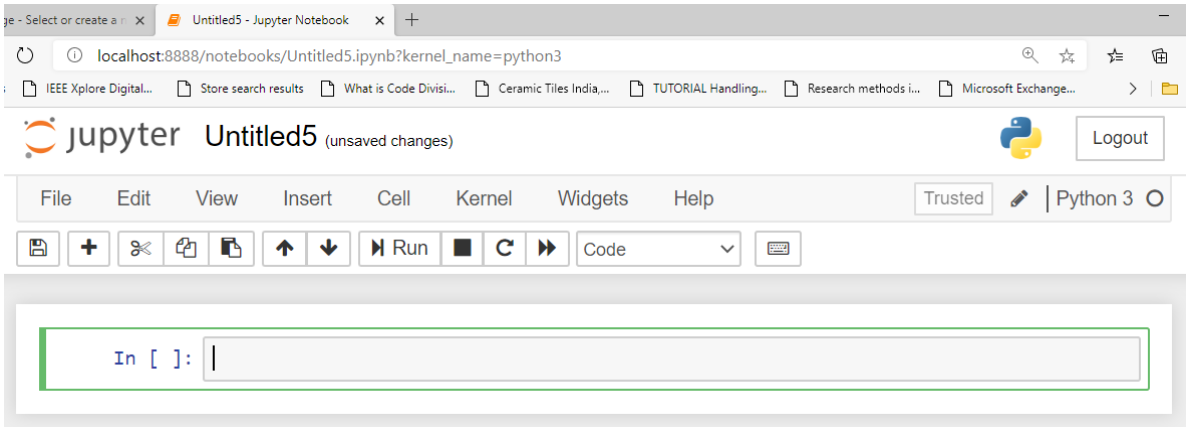
1. On Navigator's Home tab, in the Applications pane on the right, scroll to the Jupyter Notebook tile and click the Install button to install Jupyter Notebook.

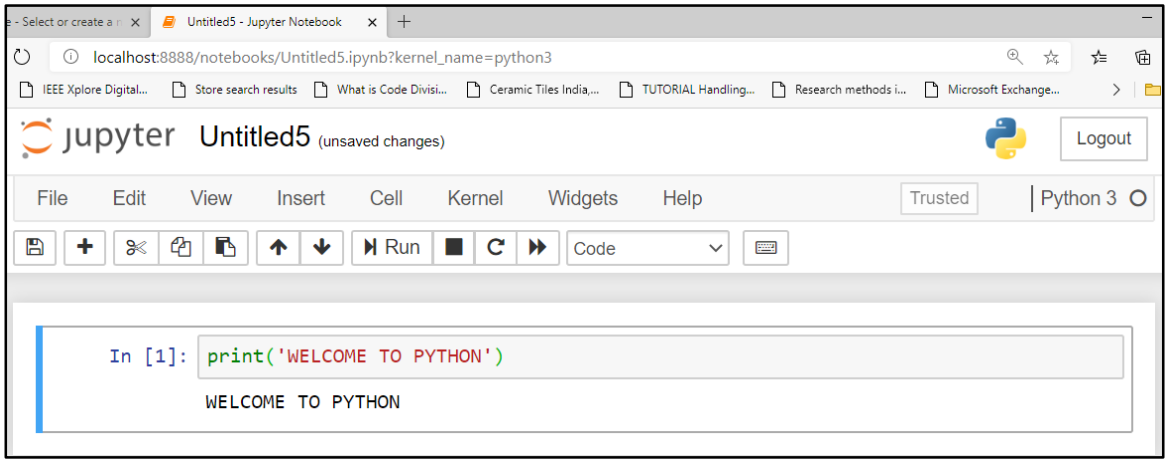
Note: If you already have Jupyter Notebook installed, you can jump right to the Launch step.

2. Launch Jupyter Notebook by clicking Jupyter Notebook's Launch button. This will launch a new browser window (or a new tab) showing the Notebook Dashboard.


Home Page - Select or create a
localhost:8888/tree
Would you like to set Microsoft Edge as your default browser?
Go to settings
jupyter
Quit
Logout
Files
Running
Clusters
Rename
Move
Upload
New
Name
Last Modified
File size
Anaconda2 8 months ago
Contacts 4 years ago
Desktop 2 minutes ago
Documents a day ago
Downloads 11 minutes ago
Favorites 4 years ago
Links 4 years ago
Music 4 days ago
OneDrive 19 days ago
Pictures 6 months ago
Roaming 4 years ago
Saved Games 4 years ago
Searches 4 years ago
Search the web and Windows
12:36 PM
7/12/2020


Upload
New
Notebook:
Python 3
Create a new notebook with Python 3
Other:
Text File
Folder
Terminal





https://docs.anaconda.com/anaconda/user-guide/getting-started/

Chapter 3 Interacting with Python

Python is a flexible and dynamic language that you can use in different ways. You can use it interactively when you simply want to test a code or a statement on a line-by-line basis or when you're exploring its features. You can use it in script mode when you want to interpret an entire file of statements or application program.

To use Python interactively, you can use either the Command Line window or the IDLE Development Environment.

Command Line Interaction: The command line is the most straightforward way to work with Python. You can easily visualize how Python works as it responds to every completed command entered on the >>> prompt.

It may not be the most preferred interaction with Python, but it is the simplest way to explore how Python works.

Starting Python

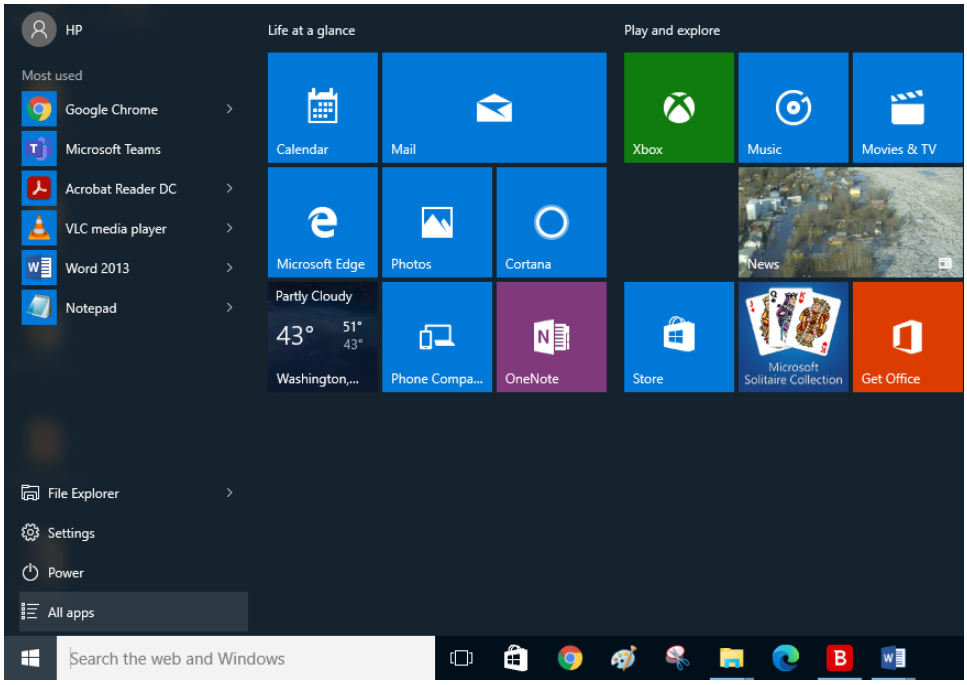
There are different ways to access Python's command line depending on the operating system installed on your machine:

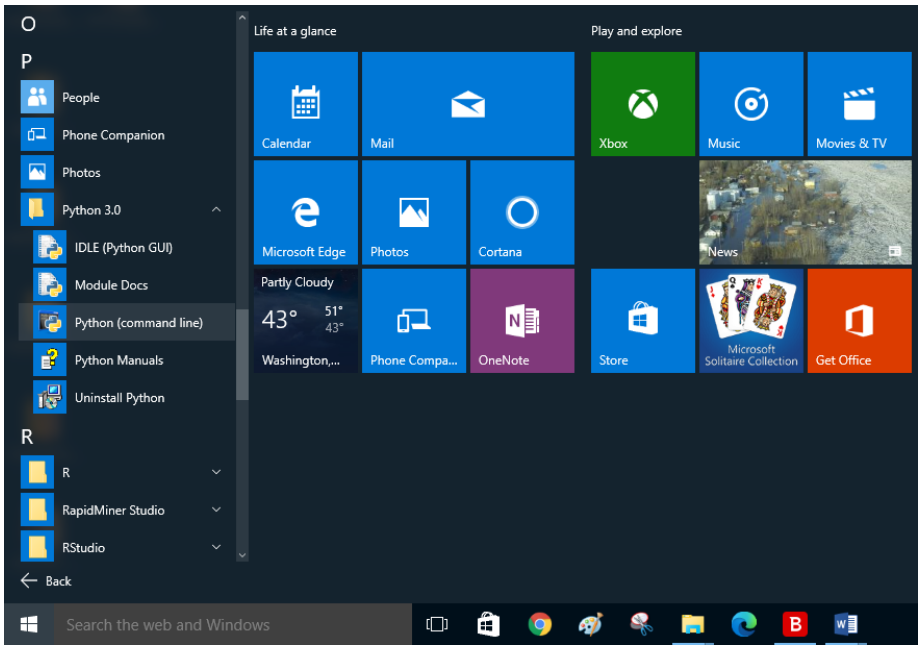
- If you're using Windows, you can start the Python command line by clicking on its icon or menu item on the Start menu.
- You may also go to the folder containing the shortcut or the installed files and click on the Python command line.
- If you're using GNU/Linux, UNIX, and Mac OS systems, you have to run the Terminal Tool and enter the Python command to start your session.

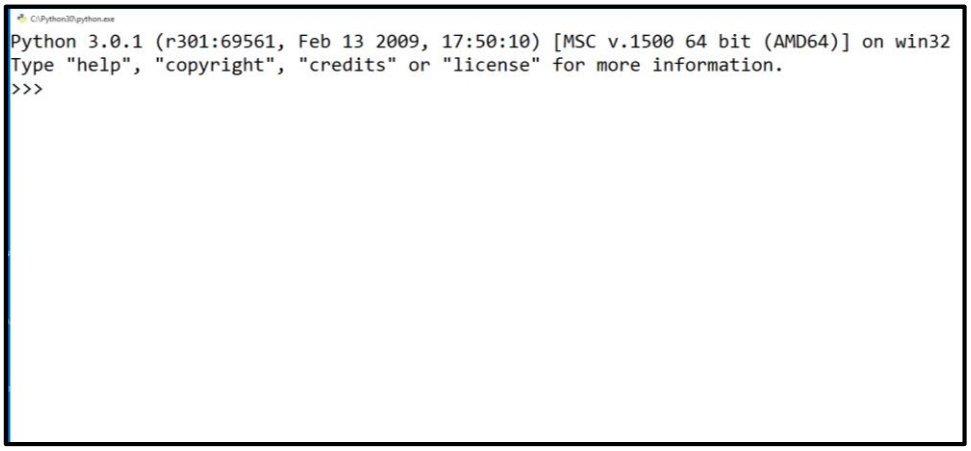
We use commands to tell the computer what to do. When you want Python to do something for you, you have to instruct it by entering commands that it is familiar with. Python will then translate these commands to instructions that your computer or device can understand and execute.

To see how Python works, you can use the print command to print the universal program "Hello, World!"

1. Open Python's command line.



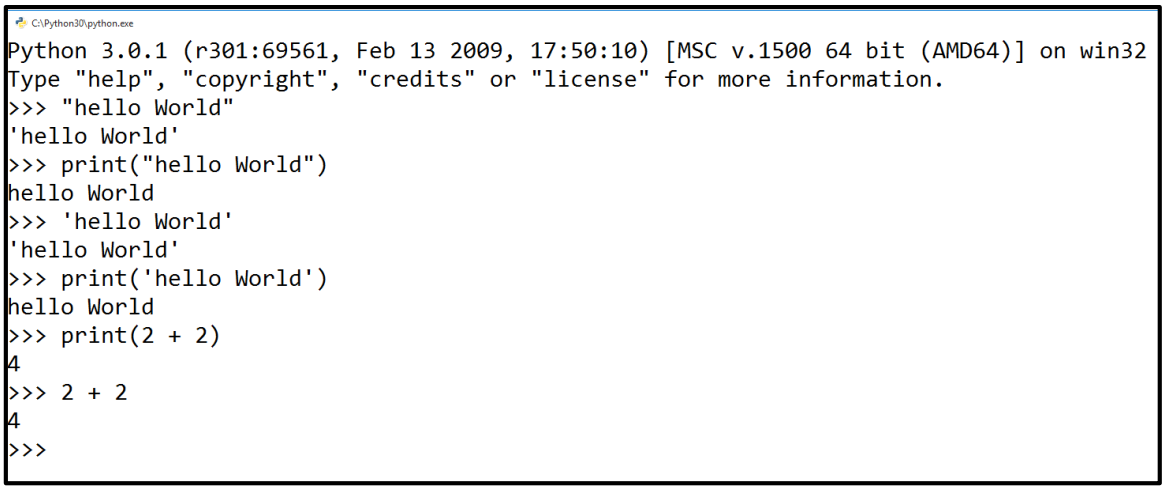




2. At the >>>prompt, type the following:

print(“Hello, World!”)

3. Press enter to tell Python that you’re done with your command. Very quickly, the command line window will display Hello, World! on the following line:



If you're just using Python interactively, you can do away with the print command entirely by just typing your statement within quotes such as "Hello, World!"

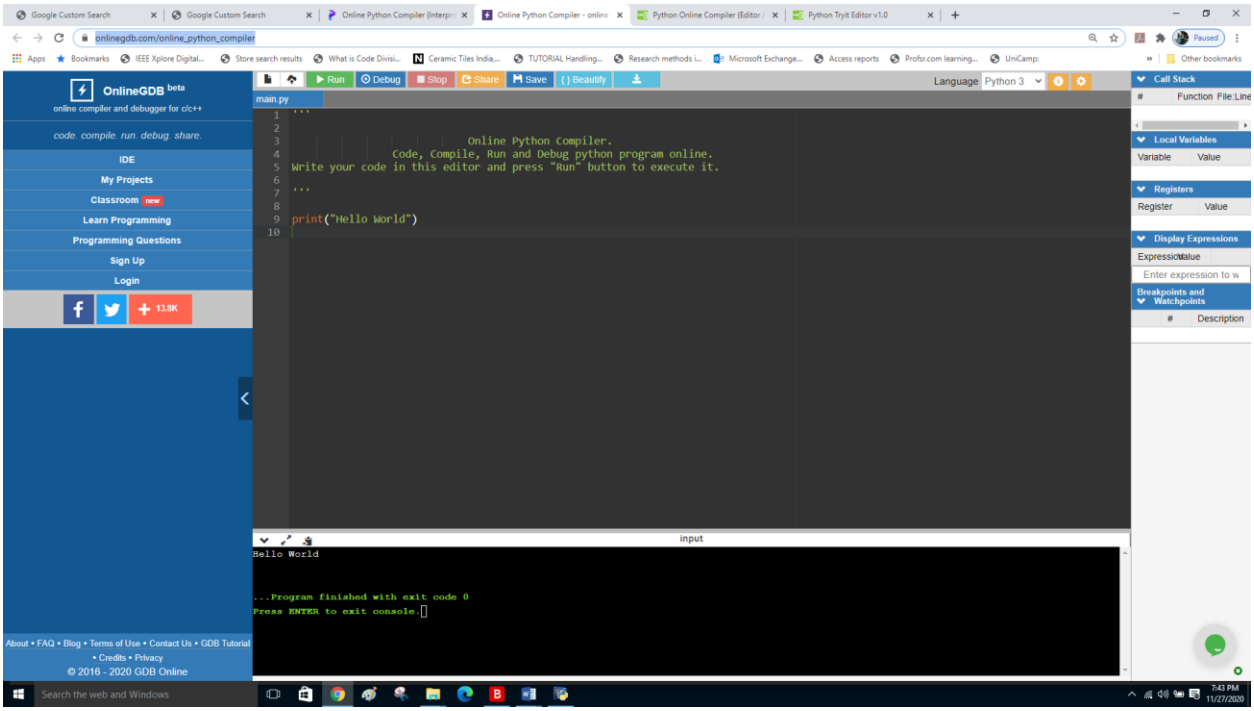
Exiting Python

To exit from Python, you can type any of these commands:

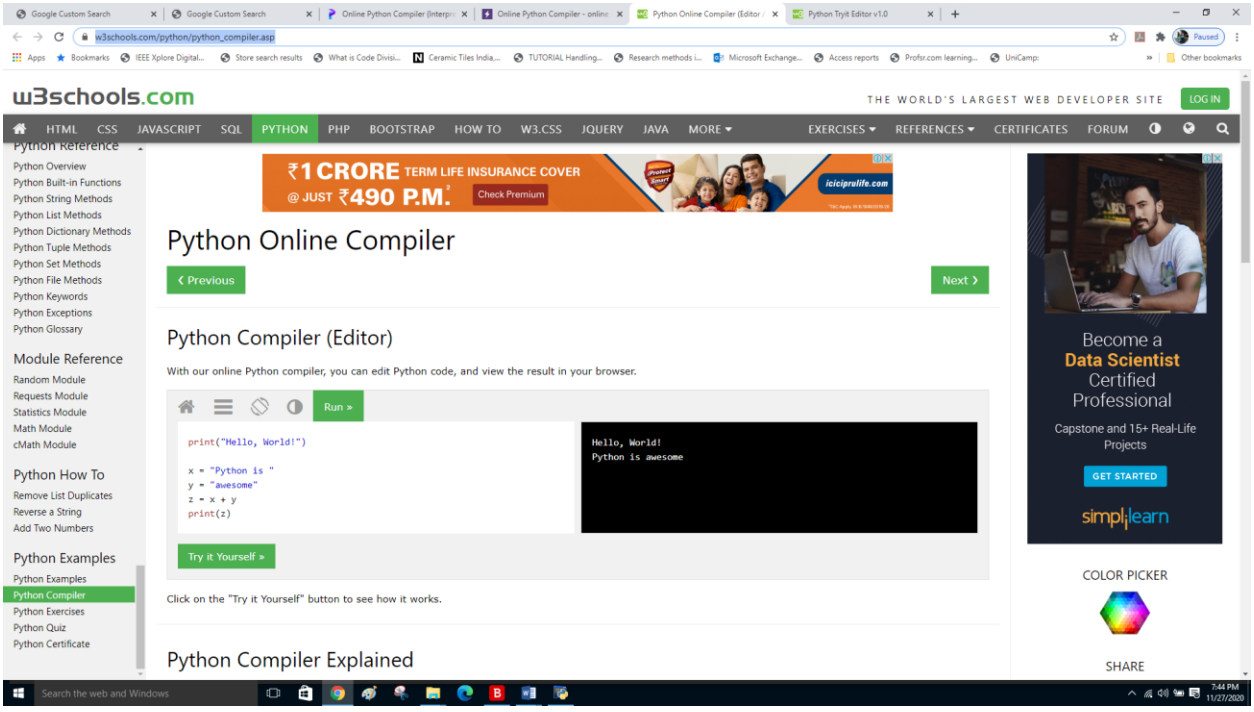
- quit()
- exit()
- Control-Z then press enter

Python Online Compilers

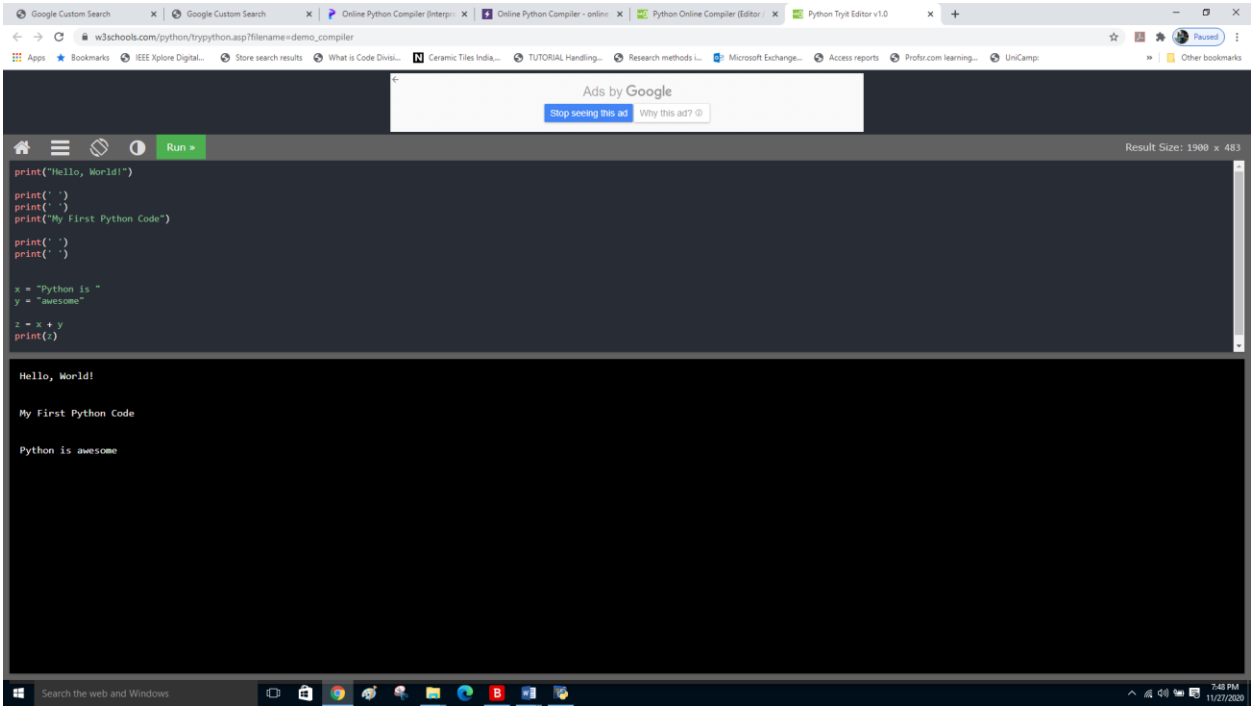
1. https://www.onlinegdb.com/online_python_compiler



2. https://www.w3schools.com/python/python_compiler.asp



3. https://www.w3schools.com/python/trypython.asp?filename=demo_compiler



Resources for Continued Learning

We recommend the following resources to support you in continuing to learn about Python development on Windows.

Online courses for learning Python

- Introduction to Python on Microsoft Learn: Try the interactive Microsoft Learn platform and earn experience points for completing this module covering the basics on how to write basic Python code, declare variables, and work with console input and output. The interactive sandbox environment makes this a great place to start for folks who don't have their Python development environment set up yet.
- Python on Pluralsight: 8 Courses, 29 Hours: The Python learning path on Pluralsight offers online courses covering a variety of topics related to Python, including a tool to measure your skill and find your gaps.
- LearnPython.org Tutorials: Get started on learning Python without needing to install or set anything up with these free interactive Python tutorials from the folks at DataCamp.
- The Python.org Tutorials: Introduces the reader informally to the basic concepts and features of the Python language and system.
- Learning Python on Lynda.com: A basic introduction to Python.

Python Identifiers

A Python identifier is a name used to identify 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 a case sensitive programming language. Thus, **Manpower** and **manpower** are two different identifiers in Python.

Here are naming conventions for Python identifiers −

- Class names start with an uppercase letter. All other identifiers start with a lowercase letter.
- Starting an identifier with a single leading underscore indicates that the identifier is private.
- Starting an identifier with two leading underscores indicates a strongly private identifier.
- If the identifier also ends with two trailing underscores, the identifier is a language-defined special name.

Reserved Words

The following list shows the Python keywords. These are reserved words and you cannot use them as constant or variable or any other identifier names. All the Python keywords contain lowercase letters only.

and	exec	not
assert	finally	or
break	for	pass
class	from	print
continue	global	raise
def	if	return
del	import	try
elif	in	while
else	is	with
except	lambda	yield

Lines and Indentation

Python provides 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. For example −

```
if True:
   print "True"
else:
   print "False"
```

However, the following block generates an error −

```
if True:
print "Answer"
print "True"
else:
print "Answer"
print "False"
```

Thus, in Python all the continuous lines indented with same number of spaces would form a block. The following example has various statement blocks −

Note – Do not try to understand the logic at this point of time. Just make sure you understood various blocks even if they are without braces.

```
#!/usr/bin/python

import sys

try:
   # open file stream
   file = open(file_name, "w")
except IOError:
   print "There was an error writing to", file_name
   sys.exit()
print "Enter '", file_finish,
print "' When finished"
while file_text != file_finish:
   file_text = raw_input("Enter text: ")
   if file_text == file_finish:
      # close the file
      file.close
      break
   file.write(file_text)
   file.write("\n")
file.close()
file_name = raw_input("Enter filename: ")
if len(file_name) == 0:
   print "Next time please enter something"
   sys.exit()
try:
   file = open(file_name, "r")
except IOError:
   print "There was an error reading file"
   sys.exit()
file_text = file.read()
file.close()
print file_text
```

Multi-Line Statements

Statements in Python typically end with a new line. Python does, however, allow the use of the line continuation character (\) to denote that the line should continue. For example −

```
total = item_one + \
        item_two + \
        item_three
```

Statements contained within the [], {}, or () brackets do not need to use the line continuation character. For example −

```
days = ['Monday', 'Tuesday', 'Wednesday',
        'Thursday', 'Friday']
```

Quotation in Python

Python accepts single ('), double (") and triple ('" or """) quotes to denote string literals, as long as the same type of quote starts and ends the string.

The triple quotes are used to span the string across multiple lines. For example, all the following are legal −

```
word = 'word'
sentence = "This is a sentence."
paragraph = """This is a paragraph. It is
made up of multiple lines and sentences."""
```

Comments in Python

A hash sign (#) that is not inside a string literal begins a comment. All characters after the # and up to the end of the physical line are part of the comment and the Python interpreter ignores them.

Live Demo

```
#!/usr/bin/python

# First comment
print "Hello, Python!" # second comment
```

This produces the following result −

```
Hello, Python!
```

You can type a comment on the same line after a statement or expression −

```
name = "Madisetti" # This is again comment
```

You can comment multiple lines as follows −

```
# This is a comment.
# This is a comment, too.
# This is a comment, too.
# I said that already.
```

Following triple-quoted string is also ignored by Python interpreter and can be used as a multiline comments:

```
'''
This is a multiline
comment.
'''
```

Using Blank Lines

A line containing only whitespace, possibly with a comment, is known as a blank line and Python totally ignores it.

In an interactive interpreter session, you must enter an empty physical line to terminate a multiline statement.

Waiting for the User

The following line of the program displays the prompt, the statement saying “Press the enter key to exit”, and waits for the user to take action −

```
#!/usr/bin/python

raw_input("\n\nPress the enter key to exit.")
```

Here, "\n\n" is used to create two new lines before displaying the actual line. Once the user presses the key, the program ends. This is a nice trick to keep a console window open until the user is done with an application.

Multiple Statements on a Single Line

The semicolon (;) allows multiple statements on the single line given that neither statement starts a new code block. Here is a sample snip using the semicolon −

```
import sys; x = 'foo'; sys.stdout.write(x + '\n')
```

Multiple Statement Groups as Suites

A group of individual statements, which make a single code block are called **suites** in Python. Compound or complex statements, such as if, while, def, and class require a header line and a suite.

Header lines begin the statement (with the keyword) and terminate with a colon (:) and are followed by one or more lines which make up the suite. For example −

```
if expression :
   suite
elif expression :
   suite
else :
   suite
```

Command Line Arguments

Many programs can be run to provide you with some basic information about how they should be run. Python enables you to do this with -h −

```
$ python -h
usage: python [option] ... [-c cmd | -m mod | file | -] [arg] ...
Options and arguments (and corresponding environment variables):
-c cmd : program passed in as string (terminates option list)
-d     : debug output from parser (also PYTHONDEBUG=x)
-E     : ignore environment variables (such as PYTHONPATH)
-h     : print this help message and exit

[ etc. ]
```

You can also program your script in such a way that it should accept various options. Command Line Arguments is an advanced topic and should be studied a bit later once you have gone through rest of the Python concepts.

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. Therefore, by assigning different data types to variables, you can store integers, decimals or characters in these variables.

Assigning Values to Variables

Python variables do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable. The equal sign (=) is used to assign values to variables.

The operand to the left of the = operator is the name of the variable and the operand to the right of the = operator is the value stored in the variable. For example −

Live Demo

```
#!/usr/bin/python

counter = 100          # An integer assignment
miles   = 1000.0       # A floating point
name    = "John"       # A string

print counter
print miles
print name
```

Here, 100, 1000.0 and "John" are the values assigned to *counter*, *miles*, and *name* variables, respectively. This produces the following result −

```
100
1000.0
John
```

Multiple Assignment

Python allows you to assign a single value to several variables simultaneously. For example −

```
a = b = c = 1
```

Here, an integer object is created with the value 1, and all three variables are assigned to the same memory location. You can also assign multiple objects to multiple variables. For example −

```
a,b,c = 1,2,"john"
```

Here, two integer objects with values 1 and 2 are assigned to variables a and b respectively, and one string object with the value "john" is assigned to the variable c.

Standard Data Types

The data stored in memory can be of many types. For example, a person's age is stored as a numeric value and his or her address is stored as alphanumeric characters. Python has various standard data types that are used to define the operations possible on them and the storage method for each of them.

Python has five standard data types −

- Numbers
- String
- List
- Tuple
- Dictionary

Python Numbers

Number data types store numeric values. Number objects are created when you assign a value to them. For example −

```
var1 = 1
var2 = 10
```

You can also delete the reference to a number object by using the del statement. The syntax of the del statement is −

```
del var1[,var2[,var3[....,varN]]]
```

You can delete a single object or multiple objects by using the del statement. For example −

```
del var
del var_a, var_b
```

Python supports four different numerical types −

- int (signed integers)
- long (long integers, they can also be represented in octal and hexadecimal)
- float (floating point real values)
- complex (complex numbers)

Examples

Here are some examples of numbers −

int	long	float	complex
10	51924361L	0.0	3.14j
100	-0x19323L	15.20	45.j
-786	0122L	-21.9	9.322e-36j
080	0xDEFABCECBDAECBFBAEl	32.3+e18	.876j
-0490	535633629843L	-90.	-.6545+0J
-0x260	-052318172735L	-32.54e100	3e+26J
0x69	-4721885298529L	70.2-E12	4.53e-7j

- Python allows you to use a lowercase l with long, but it is recommended that you use only an uppercase L to avoid confusion with the number 1. Python displays long integers with an uppercase L.
- A complex number consists of an ordered pair of real floating-point numbers denoted by x + yj, where x and y are the real numbers and j is the imaginary unit.

Python Strings

Strings in Python are identified as a contiguous set of characters represented in the quotation marks. Python allows for either pairs of single or double quotes. Subsets of strings can be taken using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the string and working their way from -1 at the end.

The plus (+) sign is the string concatenation operator and the asterisk (*) is the repetition operator. For example −

Live Demo

```
#!/usr/bin/python

str = 'Hello World!'

print str          # Prints complete string
print str[0]       # Prints first character of the string
print str[2:5]     # Prints characters starting from 3rd to 5th
print str[2:]      # Prints string starting from 3rd character
print str * 2      # Prints string two times
print str + "TEST" # Prints concatenated string
```

This will produce the following result −

```
Hello World!
H
llo
llo World!
Hello World!Hello World!
Hello World!TEST
```

Python Lists

Lists are the most versatile of Python's compound data types. A list contains items separated by commas and enclosed within square brackets ([]). To some extent, lists are similar to arrays in C. One difference between them is that all the items belonging to a list can be of different data type.

The values stored in a list can be accessed using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end -1. The plus (+) sign is the list concatenation operator, and the asterisk (*) is the repetition operator. For example −

```
#!/usr/bin/python

list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
tinylist = [123, 'john']

print list          # Prints complete list
print list[0]       # Prints first element of the list
print list[1:3]     # Prints elements starting from 2nd till 3rd
print list[2:]      # Prints elements starting from 3rd element
print tinylist * 2  # Prints list two times
print list + tinylist # Prints concatenated lists
```

This produce the following result −

```
['abcd', 786, 2.23, 'john', 70.2]
abcd
[786, 2.23]
[2.23, 'john', 70.2]
[123, 'john', 123, 'john']
['abcd', 786, 2.23, 'john', 70.2, 123, 'john']
```

Python Tuples

A tuple is another sequence data type that is similar to the list. A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parentheses.

The main differences between lists and tuples are: Lists are enclosed in brackets ([ ]) and their elements and size can be changed, while tuples are enclosed in parentheses (( )) and cannot be updated. Tuples can be thought of as **read-only** lists. For example −

Live Demo

```
#!/usr/bin/python

tuple = ( 'abcd', 786 , 2.23, 'john', 70.2  )
tinytuple = (123, 'john')

print tuple               # Prints the complete tuple
print tuple[0]            # Prints first element of the tuple
print tuple[1:3]          # Prints elements of the tuple starting from 2nd till 3rd
print tuple[2:]           # Prints elements of the tuple starting from 3rd element
print tinytuple * 2       # Prints the contents of the tuple twice
print tuple + tinytuple   # Prints concatenated tuples
```

This produce the following result −

```
('abcd', 786, 2.23, 'john', 70.2)
abcd
(786, 2.23)
(2.23, 'john', 70.2)
(123, 'john', 123, 'john')
('abcd', 786, 2.23, 'john', 70.2, 123, 'john')
```

The following code is invalid with tuple, because we attempted to update a tuple, which is not allowed. Similar case is possible with lists −

```
#!/usr/bin/python

tuple = ( 'abcd', 786 , 2.23, 'john', 70.2  )
list = [ 'abcd', 786 , 2.23, 'john', 70.2  ]
tuple[2] = 1000    # Invalid syntax with tuple
list[2] = 1000     # Valid syntax with list
```

Operators are the constructs which can manipulate the value of operands.

Consider the expression 4 + 5 = 9. Here, 4 and 5 are called operands and + is called operator.

Types of Operator

Python language supports the following types of operators.

- Arithmetic Operators
- Comparison (Relational) Operators
- Assignment Operators
- Logical Operators
- Bitwise Operators
- Membership Operators
- Identity Operators

Let us have a look on all operators one by one.

Python Arithmetic Operators

Assume variable a holds 10 and variable b holds 20, then −

[Show Example]

Operator	Description	Example
+ Addition	Adds values on either side of the operator.	a + b = 30
- Subtraction	Subtracts right hand operand from left hand operand.	a – b = -10
* Multiplication	Multiplies values on either side of the operator	a * b = 200
/ Division	Divides left hand operand by right hand operand	b / a = 2
% Modulus	Divides left hand operand by right hand operand and returns remainder	b % a = 0
** Exponent	Performs exponential (power) calculation on operators	a**b =10 to the power 20
//	Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed. But if one of the operands is negative, the result is floored, i.e., rounded away from zero (towards negative infinity) −	9//2 = 4 and 9.0//2.0 = 4.0, -11//3 = -4, -11.0//3 = -4.0

Python Comparison Operators

These operators compare the values on either sides of them and decide the relation among them. They are also called Relational operators.

Assume variable a holds 10 and variable b holds 20, then −

[Show Example]

Operator	Description	Example
==	If the values of two operands are equal, then the condition becomes true.	(a == b) is not true.
!=	If values of two operands are not equal, then condition becomes true.	(a != b) is true.
<>	If values of two operands are not equal, then condition becomes true.	(a <> b) is true. This is similar to != operator.
>	If the value of left operand is greater than the value of right operand, then condition becomes true.	(a > b) is not true.
<	If the value of left operand is less than the value of right operand, then condition becomes true.	(a < b) is true.
>=	If the value of left operand is greater than or equal to the value of right operand, then condition becomes true.	(a >= b) is not true.
<=	If the value of left operand is less than or equal to the value of right operand, then condition becomes true.	(a <= b) is true.

Python Assignment Operators

Assume variable a holds 10 and variable b holds 20, then −

[Show Example]

Operator	Description	Example
=	Assigns values from right side operands to left side operand	c = a + b assigns value of a + b into c
+= Add AND	It adds right operand to the left operand and assign the result to left operand	c += a is equivalent to c = c + a
-= Subtract AND	It subtracts right operand from the left operand and assign the result to left operand	c -= a is equivalent to c = c - a
*= Multiply AND	It multiplies right operand with the left operand and assign the result to left operand	c *= a is equivalent to c = c * a
/= Divide AND	It divides left operand with the right operand and assign the result to left operand	c /= a is equivalent to c = c / a
%= Modulus AND	It takes modulus using two operands and assign the result to left operand	c %= a is equivalent to c = c % a
**= Exponent AND	Performs exponential (power) calculation on operators and assign value to the left operand	c **= a is equivalent to c = c ** a
//= Floor Division	It performs floor division on operators and assign value to the left operand	c //= a is equivalent to c = c // a

Python Bitwise Operators

Bitwise operator works on bits and performs bit by bit operation. Assume if a = 60; and b = 13; Now in the binary format their values will be 0011 1100 and 0000 1101 respectively. Following table lists out the bitwise operators supported by Python language with an example each in those, we use the above two variables (a and b) as operands −

a = 0011 1100

b = 0000 1101

---

a&b = 0000 1100

a|b = 0011 1101

a^b = 0011 0001

~a = 1100 0011

There are following Bitwise operators supported by Python language

[Show Example]

Operator	Description	Example
& Binary AND	Operator copies a bit to the result if it exists in both operands	(a & b) (means 0000 1100)
| Binary OR	It copies a bit if it exists in either operand.	(a | b) = 61 (means 0011 1101)
^ Binary XOR	It copies the bit if it is set in one operand but not both.	(a ^ b) = 49 (means 0011 0001)
~ Binary Ones Complement	It is unary and has the effect of 'flipping' bits.	(~a) = -61 (means 1100 0011 in 2's complement form due to a signed binary number.
<< Binary Left Shift	The left operands value is moved left by the number of bits specified by the right operand.	a << 2 = 240 (means 1111 0000)
>> Binary Right Shift	The left operands value is moved right by the number of bits specified by the right operand.	a >> 2 = 15 (means 0000 1111)

Python Logical Operators

There are following logical operators supported by Python language. Assume variable a holds 10 and variable b holds 20 then

[Show Example]

Operator	Description	Example
and Logical AND	If both the operands are true then condition becomes true.	(a and b) is true.
or Logical OR	If any of the two operands are non-zero then condition becomes true.	(a or b) is true.
not Logical NOT	Used to reverse the logical state of its operand.	Not(a and b) is false.

Python Membership Operators

Python's membership operators test for membership in a sequence, such as strings, lists, or tuples. There are two membership operators as explained below –

[Show Example]

Operator	Description	Example
in	Evaluates to true if it finds a variable in the specified sequence and false otherwise.	x in y, here in results in a 1 if x is a member of sequence y.
not in	Evaluates to true if it does not finds a variable in the specified sequence and false otherwise.	x not in y, here not in results in a 1 if x is not a member of sequence y.

Python Identity Operators

Identity operators compare the memory locations of two objects. There are two Identity operators explained below −

[Show Example]

Operator	Description	Example
is	Evaluates to true if the variables on either side of the operator point to the same object and false otherwise.	x is y, here **is** results in 1 if id(x) equals id(y).
is not	Evaluates to false if the variables on either side of the operator point to the same object and true otherwise.	x is not y, here **is not** results in 1 if id(x) is not equal to id(y).

Python Operators Precedence

The following table lists all operators from highest precedence to lowest.

[Show Example]

Sr.No.	Operator & Description
1	** Exponentiation (raise to the power)
2	**~ + -** Complement, unary plus and minus (method names for the last two are +@ and -@)
3	***/ % //*** Multiply, divide, modulo and floor division
4	**+ -** Addition and subtraction
5	**>> <<** Right and left bitwise shift
6	**&** Bitwise 'AND'
7	**^ |** Bitwise exclusive `OR' and regular `OR'

8	**<= < > >=** Comparison operators
9	**<> == !=** Equality operators
10	**= %= /= //= -= += *= **=** Assignment operators
11	**is is not** Identity operators
12	**in not in** Membership operators
13	**not or and** Logical operators

Sample Python Programs

```
import os

path=os.getcwd()

print('PATH =',path)
```

```
PATH = C:\Users\hp
```

Variables and Types

Python is completely object oriented, and not "statically typed". You do not need to declare variables before using them, or declare their type. Every variable in Python is an object.

Numbers

Python supports two types of numbers - integers and floating point numbers. (It also supports complex numbers, which will not be explained in this tutorial).

To define an integer, use the following syntax:

```
myint = 7
print(myint)
```

Strings

Strings are defined either with a single quote or a double quotes.

```
mystring = 'hello'
print(mystring)
mystring = "hello"
print(mystring)
```

Simple operators can be executed on numbers and strings:

```
one = 1
two = 2
three = one + two
print(three)

hello = "hello"
```

```
world = "world"
helloworld = hello + " " + world
print(helloworld)
```

Assignments can be done on more than one variable "simultaneously" on the same line like this

```
a, b = 3, 4
print(a,b)
```

Exercise

The target of this exercise is to create a string, an integer, and a floating point number. The string should be named mystring and should contain the word "hello". The floating point number should be named myfloat and should contain the number 10.0, and the integer should be named myint and should contain the number 20.

```
# change this code
mystring = None
myfloat = None
myint = None

# testing code
if mystring == "hello":
    print("String: %s" % mystring)
if isinstance(myfloat, float) and myfloat == 10.0:
    print("Float: %f" % myfloat)
if isinstance(myint, int) and myint == 20:
    print("Integer: %d" % myint)
```

```python
# change this code
mystring = 'hello'
myfloat = None
myint = None

# testing code
if mystring == "hello":
    print("String: %s" % mystring)
if isinstance(myfloat, float) and myfloat == 10.0:
    print("Float: %f" % myfloat)
if isinstance(myint, int) and myint == 20:
    print("Integer: %d" % myint)
```

```
String: hello
```

```python
# change this code
mystring = None
myfloat = None
myint = 20

# testing code
if mystring == "hello":
    print("String: %s" % mystring)
if isinstance(myfloat, float) and myfloat == 10.0:
    print("Float: %f" % myfloat)
if isinstance(myint, int) and myint == 20:
    print("Integer: %d" % myint)
```

```
Integer: 20
```

https://docs.microsoft.com/en-us/windows/python/beginners#:~:text=To%20install%20Python%20using%20the%20Microsoft%20Store%3A%201,to%20install%20and%20manage%20additional%20packages%20...%20

Scikit-learn is a Python module integrating a wide range of state-of-the-art machine learning algorithms for medium-scale supervised and unsupervised problems. This package focuses on bringing machine learning to non-specialists using a general-purpose high-level language. Emphasis is put on ease of use, performance, documentation, and API consistency. It has minimal dependencies and is distributed under the simplified BSD license, encouraging its use in both academic and commercial settings. Source code, binaries, and documentation can be downloaded from http://scikit-learn.sourceforge.net.

Scikit-learn harnesses this rich environment to provide state-of-the-art implementations of many well-known machine learning algorithms, while maintaining an easy-to-use interface tightly integrated with the Python language. This answers the growing need for statistical data analysis by non-specialists in the software and web industries, as well as in fields outside of computer-science, such as biology or physics. *Scikit-learn* differs from other machine learning toolboxes in Python for various reasons: *i)* it is distributed under the BSD license *ii)* it incorporates compiled code for efficiency, unlike MDP (Zito et al., 2008) and pybrain (Schaul et al., 2010), *iii)* it depends only on numpy and scipy to facilitate easy distribution.

Numpy: the base data structure used for data and model parameters. Input data is presented as numpy arrays, thus integrating seamlessly with other scientific Python libraries. Numpy's view based memory model limits copies, even when binding with compiled code (Van der Walt et al., 2011). It also provides basic arithmetic operations.

Scipy: Efficient algorithms for linear algebra, sparse matrix representation, special functions and basic statistical functions. Scipy has bindings for many Fortran-based standard numerical packages, such as LAPACK. This is important for ease of installation and portability, as providing libraries around Fortran code can prove challenging on various platforms.

Cython: a language for combining C in Python. Cython makes it easy to reach the performance of compiled languages with Python-like syntax and high-level operations. It is also used to bind compiled libraries, eliminating the boilerplate code of Python/C extensions.

https://archive.ics.uci.edu/ml/machine-learning-databases/undocumented/connectionist-bench/sonar/sonar.all-data

References:

1) Andrew Johansen, “Python: The Ultimate Beginners Guide”, 2016.

2) Mark Lutz, “Learning Python”, Third Edition, Oreilly Series, 2008.

3) Tim Hall and J-P Stacey, “Python 3 for Absolute Beginners”, Apress, 2009.

4) Dave Brueck and Stephen Tanner, “Python 2.1 Bible”, Hungry Minds Inc, 2001.

5) “Python for Data Analysis: Data Wrangling with Pandas, NumPy,and IPython, Second Edition, Oreilly, 2018.

6) Wes McKinney and the Pandas Development Team, “pandas: powerful Python data analysis Toolkit Release 1.0.5, 2020.

7) Kent D.Lee, “Python Programming Fundamentals”, Second Edition, Springer publications, 2014.

If you are not sure what type a value has, the interpreter can tell you.

```
>>> type('Hello, World!')
<type 'str'>
>>> type(17)
<type 'int'>
```

```
>>> type(3.2)
<type 'float'>
```

```
>>> type('17')
<type 'str'>
>>> type('3.2')
<type 'str'>
```

They're strings.

When you type a large integer, you might be tempted to use commas between groups of three digits, as in `1,000,000`. This is not a legal integer in Python, but it is legal:

```
>>> 1,000,000
(1, 0, 0)
```

Well, that's not what we expected at all! Python interprets `1,000,000` as a comma-separated sequence of integers. This is the first example we have seen of a semantic error: the code runs without producing an error message, but it doesn't do the "right" thing.

An **assignment statement** creates new variables and gives them values:

```
>>> message = 'And now for something completely different'
>>> n = 17
>>> pi = 3.1415926535897932
```

This example makes three assignments. The first assigns a string to a new variable named `message`; the second gives the integer `17` to `n`; the third assigns the (approximate) value of π to `pi`.

The type of a variable is the type of the value it refers to.

```
>>> type(message)
<type 'str'>
>>> type(n)
<type 'int'>
>>> type(pi)
<type 'float'>
```

Variable names and keywords

Variable names can be arbitrarily long. They can contain both letters and numbers, but they have to begin with a letter. It is legal to use uppercase letters, but it is a good idea to begin variable names with a lowercase letter (you'll see why later).

The underscore character, `_`, can appear in a name. It is often used in names with multiple words, such as `my_name` or `airspeed_of_unladen_swallow`.

If you give a variable an illegal name, you get a syntax error:

```
>>> 76trombones = 'big parade'
SyntaxError: invalid syntax
>>> more@ = 1000000
SyntaxError: invalid syntax
>>> class = 'Advanced Theoretical Zymurgy'
SyntaxError: invalid syntax
```

`76trombones` is illegal because it does not begin with a letter. `more@` is illegal because it contains an illegal character, `@`. But what's wrong with `class`?

It turns out that `class` is one of Python's **keywords**. The interpreter uses keywords to recognize the structure of the program, and they cannot be used as variable names.

Python 2 has 31 keywords:

```
and       del       from      not       while
as        elif      global    or        with
assert    else      if        pass      yield
break     except    import    print
class     exec      in        raise
continue  finally   is        return
def       for       lambda    try
```

In Python 3, `exec` is no longer a keyword, but `nonlocal` is.

You might want to keep this list handy. If the interpreter complains about one of your variable names and you don't know why, see if it is on this list.

Operators and operands

Operators are special symbols that represent computations like addition and multiplication. The values the operator is applied to are called **operands**.

The operators `+`, `-`, `*`, `/` and `**` perform addition, subtraction, multiplication, division and exponentiation, as in the following examples:

```
20+32    hour-1    hour*60+minute    minute/60    5**2    (5+9)*(15-7)
```

In some other languages, `^` is used for exponentiation, but in Python it is a bitwise operator called XOR. I won't cover bitwise operators in this book, but you can read about them at `http://wiki.python.org/moin/BitwiseOperators`.

In Python 2, the division operator might not do what you expect:

```
>>> minute = 59
>>> minute/60
0
```

The value of `minute` is 59, and in conventional arithmetic 59 divided by 60 is 0.98333, not 0. The reason for the discrepancy is that Python is performing **floor division**. When both of the operands are integers, the result is also an integer; floor division chops off the fraction part, so in this example it rounds down to zero.

In Python 3, the result of this division is a `float`. The new operator `//` performs floor division.

If either of the operands is a floating-point number, Python performs floating-point division, and the result is a `float`:

```
>>> minute/60.0
0.98333333333333328
```

Expressions and statements

An **expression** is a combination of values, variables, and operators. A value all by itself is considered an expression, and so is a variable, so the following are all legal expressions (assuming that the variable `x` has been assigned a value):

```
17
x
x + 17
```

A **statement** is a unit of code that the Python interpreter can execute. We have seen two kinds of statement: print and assignment.

Technically an expression is also a statement, but it is probably simpler to think of them as different things. The important difference is that an expression has a value; a statement does not.

Interactive mode and script mode

One of the benefits of working with an interpreted language is that you can test bits of code in interactive mode before you put them in a script. But there are differences between interactive mode and script mode that can be confusing.

For example, if you are using Python as a calculator, you might type

```
>>> miles = 26.2
>>> miles * 1.61
42.182
```

The first line assigns a value to `miles`, but it has no visible effect. The second line is an expression, so the interpreter evaluates it and displays the result. So we learn that a marathon is about 42 kilometers.

But if you type the same code into a script and run it, you get no output at all. In script mode an expression, all by itself, has no visible effect. Python actually evaluates the expression, but it doesn't display the value unless you tell it to:

```
miles = 26.2
print miles * 1.61
```

This behavior can be confusing at first.

A script usually contains a sequence of statements. If there is more than one statement, the results appear one at a time as the statements execute.

For example, the script:

```
print 1
x = 2
print x
```

produces the output

```
1
2
```

The assignment statement produces no output.

Order of operations

When more than one operator appears in an expression, the order of evaluation depends on the **rules of precedence**. For mathematical operators, Python follows mathematical convention. The acronym **PEMDAS** is a useful way to remember the rules:

- **P**arentheses have the highest precedence and can be used to force an expression to evaluate in the order you want. Since expressions in parentheses are evaluated first, `2 * (3-1)` is 4, and `(1+1)**(5-2)` is 8. You can also use parentheses to make an expression easier to read, as in `(minute * 100) / 60`, even if it doesn't change the result.
- **E**xponentiation has the next highest precedence, so `2**1+1` is 3, not 4, and `3*1**3` is 3, not 27.
- **M**ultiplication and **D**ivision have the same precedence, which is higher than **A**ddition and **S**ubtraction, which also have the same precedence. So `2*3-1` is 5, not 4, and `6+4/2` is 8, not 5.
- Operators with the same precedence are evaluated from left to right (except exponentiation). So in the expression `degrees / 2 * pi`, the division happens first and the result is multiplied by `pi`. To divide by 2π, you can use parentheses or write `degrees / 2 / pi`.

String operations

In general, you can't perform mathematical operations on strings, even if the strings look like numbers, so the following are illegal:

```
'2'-'1'    'eggs'/'easy'    'third'*'a charm'
```

The + operator works with strings, but it might not do what you expect: it performs **concatenation**, which means joining the strings by linking them end-to-end. For example:

```
first = 'throat'
second = 'warbler'
print first + second
```

The output of this program is `throatwarbler`.

The * operator also works on strings; it performs repetition. For example, `'Spam'*3` is `'SpamSpamSpam'`. If one of the operands is a string, the other has to be an integer.

This use of + and * makes sense by analogy with addition and multiplication. Just as `4*3` is equivalent to `4+4+4`, we expect `'Spam'*3` to be the same as `'Spam'+'Spam'+'Spam'`, and it is. On the other hand, there is a significant way in which string concatenation and repetition are different from integer addition and multiplication. Can you think of a property that addition has that string concatenation does not?

Comments

As programs get bigger and more complicated, they get more difficult to read. Formal languages are dense, and it is often difficult to look at a piece of code and figure out what it is doing, or why.

For this reason, it is a good idea to add notes to your programs to explain in natural language what the program is doing. These notes are called **comments**, and they start with the # symbol:

```
# compute the percentage of the hour that has elapsed
percentage = (minute * 100) / 60
```

In this case, the comment appears on a line by itself. You can also put comments at the end of a line:

```
percentage = (minute * 100) / 60     # percentage of an hour
```

Everything from the # to the end of the line is ignored—it has no effect on the program.

Comments are most useful when they document non-obvious features of the code. It is reasonable to assume that the reader can figure out *what* the code does; it is much more useful to explain *why*.

This comment is redundant with the code and useless:

```
v = 5      # assign 5 to v
```

This comment contains useful information that is not in the code:

```
v = 5      # velocity in meters/second.
```

Good variable names can reduce the need for comments, but long names can make complex expressions hard to read, so there is a tradeoff.

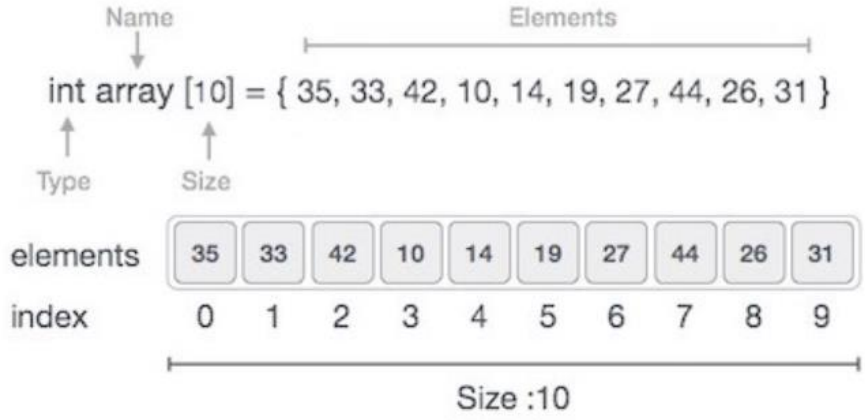
Python - Arrays

Array is a container which can hold a fix number of items and these items should be of the same type. Most of the data structures make use of arrays to implement their algorithms. Following are the important terms to understand the concept of Array.

- **Element**− Each item stored in an array is called an element.
- **Index** − Each location of an element in an array has a numerical index, which is used to identify the element.

Array Representation

Arrays can be declared in various ways in different languages. Below is an illustration.



As per the above illustration, following are the important points to be considered.

- Index starts with 0.
- Array length is 10 which means it can store 10 elements.
- Each element can be accessed via its index. For example, we can fetch an element at index 6 as 9.

Basic Operations

Following are the basic operations supported by an array.

- **Traverse** − print all the array elements one by one.
- **Insertion** − Adds an element at the given index.
- **Deletion** − Deletes an element at the given index.
- **Search** − Searches an element using the given index or by the value.
- **Update** − Updates an element at the given index.

Array is created in Python by importing array module to the python program. Then the array is declared as shown eblow.

```
from array import *

arrayName = array(typecode, [Initializers])
```

Typecode are the codes that are used to define the type of value the array will hold. Some common typecodes used are:

Typecode	Value
b	Represents signed integer of size 1 byte/td>
B	Represents unsigned integer of size 1 byte
c	Represents character of size 1 byte
i	Represents signed integer of size 2 bytes
I	Represents unsigned integer of size 2 bytes
f	Represents floating point of size 4 bytes
d	Represents floating point of size 8 bytes

Before lookign at various array operations lets create and print an array using python.

The below code creates an array named array1.

```
from array import *

array1 = array('i', [10,20,30,40,50])

for x in array1:
 print(x)
```

When we compile and execute the above program, it produces the following result –

Output

```
10
20
30
40
50
```

Accessing Array Element

We can access each element of an array using the index of the element. The below code shows how

```
from array import *

array1 = array('i', [10,20,30,40,50])

print (array1[0])

print (array1[2])
```

When we compile and execute the above program, it produces the following result – which shows the element is inserted at index position 1.

Output

```
10
30
```

Insertion Operation

Insert operation is to insert one or more data elements into an array. Based on the requirement, a new element can be added at the beginning, end, or any given index of array.

Here, we add a data element at the middle of the array using the python in-built insert() method.

```
from array import *

array1 = array('i', [10,20,30,40,50])

array1.insert(1,60)

for x in array1:
 print(x)
```

When we compile and execute the above program, it produces the following result which shows the element is inserted at index position 1.

Output

```
10
60
20
30
40
50
```

Deletion Operation

Deletion refers to removing an existing element from the array and re-organizing all elements of an array.

Here, we remove a data element at the middle of the array using the python in-built remove() method.

```
from array import *

array1 = array('i', [10,20,30,40,50])

array1.remove(40)

for x in array1:
 print(x)
```

When we compile and execute the above program, it produces the following result which shows the element is removed form the array.

Output

```
10
20
30
50
```

Search Operation

You can perform a search for an array element based on its value or its index.

Here, we search a data element using the python in-built index() method.

```
from array import *

array1 = array('i', [10,20,30,40,50])

print (array1.index(40))
```

When we compile and execute the above program, it produces the following result which shows the index of the element. If the value is not present in the array then th eprogram returns an error.

Output

```
3
```