Gizmofacts

Python Basics Overview

Kaustav Ghosh Dastidar

www.gizmofacts.com

Python Basics Overview

Preface

This book is written to help you to learn Python programming quickly and effectively. If you are new to programming, this book will help you straightforwardly understand complex concepts. Each concept is demonstrated with carefully chosen examples to help you gain a better understanding of the language from the basics to the advanced level.

If you are a skilled programmer, this book provides a solid foundation from which to explore your knowledge of Python to the next levels and become a top-tier software engineer.

Each chapter will walk you through simple and practical approaches to learning the Python programming language.

Python is consistently ranked as one of the world's most popular programming languages. It is a free, open-source programming language with a large community and extensive support modules, as well as easy integration with web services, user-friendly data structures, and graphical user interface (GUI)-based desktop applications. It is a popular programming language that is used in machine learning and deep learning applications.

So, if you are just getting started with programming, Python could be a great choice. Now a days Python is taught as the primary programming language in a wide range of schools, colleges, and universities.

Why You should Learn Python?

If you're thinking about learning Python but aren't sure why here are ten reasons why you should.

Data Science and Machine learning

This is the single most important reason why many programmers will be learning Python in 2023.

learning Python makes sense because it is quickly becoming the most popular programming language, and there are powerful AI, Data Science, and Machine Learning APIs and libraries available.

Python for Hacking

In the cyber security industry, programming is one of the most important ethical hacking tools. After learning Python for cyber security, you will be able to identify any potential threat and obtain additional cyber security training. This programming language comes in handy for attack vectors, security flaws, and common attacks.

Python for Mobile Application Development

Kivy and Beware are two Python frameworks for developing mobile applications.
Python also has libraries and frameworks that allow you to write code once and run it on multiple platforms (i.e. Android, iOS). This is known as cross-platform development.

Python for Graphics Design

As a developer, you can use Python to create web applications, games, and a variety of GUI tools, among other things. Python Graphics User Interface (GUI) is extremely useful for a wide range of projects.

You can use these technologies to create a one-of-a-kind, aesthetically pleasing, visually appealing, highly interactive environment for your projects, as well as provide users with other wonderful features.

Web scraping with python!

Consider the following scenario: you need to quickly retrieve a large amount of data from websites. What if you didn't go to each website and manually collect the data?

Web scraping is the solution. Web scraping simply makes this process easier and faster.

Python, as an object-oriented language, is one of the simplest to learn. Classes and objects in Python are significantly easier to use than in any other language. Furthermore, many libraries exist that make developing a web scraping tool in Python a breeze.

Python Basics Overview is essentially a multi-sensory learning experience that will assist you in becoming a true Python programmer!

With Warm Regards,

Kaustav Ghosh Dastidar

Author & Founder of Gizmofacts

About The Author

The man behind **Gizmofacts, Kaustav Ghosh Dastidar** holds a Master Degree in Computer Science. He is usually turned on (metaphorically) by technology and gadgets. Born and raised in a quintessential middle class family he has been well aware of the ignorance the mass harbours about technology. Through Gizmofacts he wants to reach out to all those people, who he believes just need a little push to get into this unique and amazing world of science and software.

Moreover, Kaustav is well aware that nurturing an interest in gadgets doesn't come cheap. Hence he wants to also be an enabler who would provide all the 'need to know' financial details of different gadgets so that people can live their dreams remaining in their budget.

To know more about tips and tricks of softwares, gizmos and mobile apps, follow him in ***Twitter***, ***Facebook*** and Google+.

You may also subscribe to Gizmofacts in ***Youtube*** for getting more information about software tips & tricks.

Disclaimer

This eBook may not be copied or distributed without permission in any way. This publication's content is offered solely for informational reasons. The usage or misuse of this eBook, as well as any financial damage incurred by individuals or property as a direct or indirect result of using this eBook, are not the author's responsibility.

We are unable to guarantee your success or outcomes in the future due to some unforeseeable risks associated with doing business online. You accept that the author is not responsible for any success or failure of your business that is related in any way to the download and use of our information and that the use of our information should be based on your due diligence.

Without the author's prior written consent, no portion of this eBook may be copied or otherwise distributed in any way, including electronically, mechanically, by photocopy, recording, or any other method

Table of Content

Python - Home

Python Tutorial

This Python tutorial has been written for the beginners to help them understand the basic to advanced concepts of Python Programming Language. After completing this tutorial, you will find yourself at a great level of expertise in Python, from where you can take yourself to the next levels to become a world class Software Engineer.

What is Python?

Python is a very popular general-purpose interpreted, interactive, object-oriented, and high-level programming language. Python is dynamically-typed and garbage-collected 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).

> Python supports multiple programming paradigms, including Procedural, Object Oriented and Functional programming language. Python design philosophy emphasizes code readability with the use of significant indentation.

Python Jobs

Today, Python is very high in demand and all the major companies are looking for great Python Programmers to develop websites, software components, and applications or to work with Data Science, AI, and ML technologies. When we are developing this tutorial in 2022, there is a high shortage of Python Programmers where as market demands more number of Python Programmers due to it's application in Machine Learning, Artificial Intelligence etc.

Today a Python Programmer with 3-5 years of experience is asking for around $150,000 annual package and this is the most demanding programming language in America. Though it can vary depending on the location of the Job. It's impossible to list all of the companies using Python, to name a few big companies are:

- Google
- Intel
- NASA
- PayPal
- Facebook
- IBM

- Amazon
- Netflix
- Pinterest
- Uber
- Many more...

So, you could be the next potential employee for any of these major companies. We have developed a great learning material for you to learn Python Programming which will help you prepare for the technical interviews and certification exams based on Python. So, start learning Python using this simple and effective tutorial from anywhere and anytime absolutely at your pace.

Why to Learn Python?

Python is consistently rated as one of the world's most popular programming languages. Python is fairly easy to learn, so if you are starting to learn any programming language then Python could be your great choice. Today various Schools, Colleges and Universities are teaching Python as their primary programming language. There are many other good reasons which makes Python as the top choice of any programmer:

- Python is Open Source which means its available free of cost.
- Python is simple and so easy to learn
- Python is versatile and can be used to create many different things.
- Python has powerful development libraries include AI, ML etc.
- Python is much in demand and ensures high salary

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.

Python Online Compiler/Interpreter

We have provided **Python Online Compiler/Interpreter** which helps you to **Edit** and **Execute** the code directly from your browser. Try to click the icon run button to run the following Python code to print conventional "Hello, World!".

> Below code box allows you to change the value of the code. Try to change the value inside **print**() and run it again to verify the result.

```
# This is my first Python program.
# This will print 'Hello, World!' as the output

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

Careers with Python

If you know Python nicely, then you have a great career ahead. Here are just a few of the career options where Python is a key skill:

- Game developer
- Web designer
- Python developer
- Full-stack developer
- Machine learning engineer
- Data scientist
- Data analyst
- Data engineer
- DevOps engineer
- Software engineer
- Many more other roles

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.

Applications of Python

The latest release of Python is 3.x. 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.

Target Audience

This tutorial has been prepared for the beginners to help them understand the basics to advanced concepts of Python programming language. After completing this tutorial, you will find yourself at a great level of expertise in Python programming, from where you can take yourself to the next levels.

Prerequisites

Although it is a beginners tutorial, we assume that the readers have a reasonable exposure to any programming environment and knowledge of basic concepts such as variables, commands, syntax, etc.

Python - Overview

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.

Python - Environment Setup

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.

- 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

Python has also been ported to the Java and .NET virtual machines

Local Environment Setup

Open a terminal window and type "python" to find out if it is already installed and which version is installed. If Python is already installed then you will get a message something like as follows:

```
$ python
Python 3.6.8 (default, Sep 10 2021, 09:13:53)
[GCC 8.5.0 20210514 (Red Hat 8.5.0-3)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

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.

Now issue the following commands:

```
$ 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.

Using Yum Command

Red Hat Enterprise Linux (RHEL 8) does not install Python 3 by default. We usually use yum command on CentOS and other related variants. The procedure for installing Python-3 on RHEL 8 is as follows:

```
$ sudo yum install python3
```

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.

Macintosh Installation

Recent Macs come with Python installed, but it may be several years out of date. See http://www.python.org/download/mac/ for instructions on getting the current version along with extra tools to support development on the Mac. For older Mac OS's before Mac OS X 10.3 (released in 2003), MacPython is available.

Jack Jansen maintains it and you can have full access to the entire documentation at his website − http://www.cwi.nl/~jack/macpython.html. You can find complete installation details for Mac OS installation.

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 −

- **n 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
6	**-c cmd** 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.

We have provided **Python Online Compiler/Interpreter** which helps you to **Edit** and **Execute** the code directly from your browser. Try to click the icon run button to run the following Python code to print conventional "Hello, World!".

> Below code box allows you to change the value of the code. Try to change the value inside print() and run it again to verify the result.

```
# This is my first Python program.
# This will print 'Hello, World!' as the output

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

Python - Basic Syntax

The Python syntax defines a set of rules that are used to create Python statements while writing a Python Program. The Python Programming Language Syntax has many similarities to Perl, C, and Java Programming Languages. However, there are some definite differences between the languages.

First Python Program

Let us execute a Python "Hello, World!" Programs in different modes of programming.

Python - Interactive Mode Programming

We can invoke a Python interpreter from command line by typing **python** at the command prompt as following −

```
$ python
Python 3.6.8 (default, Sep 10 2021, 09:13:53)
[GCC 8.5.0 20210514 (Red Hat 8.5.0-3)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Here >>> denotes a Python Command Prompt where you can type your commands. Let's type the following text at the Python prompt and press the Enter −

```
>>> print ("Hello, World!")
```

If you are running older version of Python, like Python 2.4.x, then you would need to use print statement without parenthesis as in **print "Hello, World!"**. However in Python version 3.x, this produces the following result −

```
Hello, World!
```

Python - Script Mode Programming

We can invoke the Python interpreter with a script parameter which begins the execution of the script and continues until the script is finished. When the script is finished, the interpreter is no longer active.

Let us write a simple Python program in a script which is simple text file. Python files have extension **.py**. Type the following source code in a **test.py** file −

```
print (“Hello, World!”)
```

We assume that you have Python interpreter path set in PATH variable. Now, let’s try to run this program as follows −

```
$ python test.py
```

This produces the following result −

```
Hello, World!
```

Let us try another way to execute a Python script. Here is the modified test.py file −

```
#!/usr/bin/python

print (“Hello, World!”)
```

We assume that you have Python interpreter available in /usr/bin directory. Now, try to run this program as follows −

```
$ chmod +x test.py     # This is to make file executable
$./test.py
```

This produces the following result −

```
Hello, World!
```

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 −

- Python 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** identifier.
- 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.

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

Python Lines and Indentation

Python programming 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 −

> 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.

```
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
```

Python 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 following statement works well in Python −

```
days = [‘Monday’, ‘Tuesday’, ‘Wednesday’,
        ‘Thursday’, ‘Friday’]
```

Quotations 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 comment is a programmer-readable explanation or annotation in the Python source code. They are added with the purpose of making the source code easier for humans to understand, and are ignored by Python interpreter

Just like most modern languages, Python supports single-line (or end-of-line) and multi-line (block) comments. Python comments are very much similar to the comments available in PHP, BASH and Perl Programming languages.

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.

```
# First comment
print (“Hello, World!”) # Second comment
```

This produces the following result −

```
Hello, World!
```

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 in Python Programs

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 in Python

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.

Python - Comments

Python comments are programmer-readable explanation or annotations in the Python source code. They are added with the purpose of making the source code easier for humans to understand, and are ignored by Python interpreter. Comments enhance the readability of the code and help the programmers to understand the code very carefully.

Just like most modern languages, Python supports single-line (or end-of-line) and multi-line (block) comments. Python comments are very much similar to the comments available in PHP, BASH and Perl Programming languages.

There are three types of comments available in Python

- Single line Comments
- Multiline Comments
- Docstring Comments

Single Line Comments

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.

Example

Following is an example of a single line comment in Python:

```
# This is a single line comment in python

print (“Hello, World!”)
```

This produces the following result −

```
Hello, World!
```

Docstring Comments

Python docstrings provide a convenient way to provide a help documentation with Python modules, functions, classes, and methods. The **docstring** is then made available via the __ doc__ attribute.

```
def add(a, b):
   “””Function to add the value of a and b”””
   return a+b

print(add.__doc__)
```

This produces the following result −

```
Function to add the value of a and b
```

Python - Variables

Python variables are the reserved memory locations used to store values with in a Python Program. This means that when you create a variable you reserve some space in the memory.

Based on the data type of a variable, Python interpreter allocates memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to Python variables, you can store integers, decimals or characters in these variables.

Creating Python Variables

Python variables do not need explicit declaration to reserve memory space or you can say to create a variable. A Python variable is created automatically when you assign a value to it. 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 −

```
counter = 100          # Creates an integer variable
miles   = 1000.0       # Creates a floating point variable
name    = “Zara Ali”   # Creates a string variable
```

Printing Python Variables

Once we create a Python variable and assign a value to it, we can print it using print() function. Following is the extension of previous example and shows how to print different variables in Python:

```
counter = 100          # Creates an integer variable
miles   = 1000.0       # Creates a floating point variable
name    = “Zara Ali”   # Creates a string variable

print (counter)
print (miles)
print (name)
```

Here, 100, 1000.0 and “Zara Ali” are the values assigned to counter, miles, and name variables, respectively. When running the above Python program, this produces the following result −

```
100
1000.0
Zara Ali
```

Delete a Variable

You can 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
```

Example

Following examples shows how we can delete a variable and if we try to use a deleted variable then Python interpreter will throw an error:

```
counter = 100
print (counter)

del counter
print (counter)
```

This will produce the following result:

```
100
Traceback (most recent call last):
  File “main.py”, line 7, in <module>
    print (counter)
NameError: name ‘counter’ is not defined
```

Multiple Assignment

Python allows you to assign a single value to several variables simultaneously which means you can create multiple variables at a time. For example −

```
a = b = c = 100

print (a)
print (b)

print (c)
```

This produces the following result:

```
100
100
100
```

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,"Zara Ali"

print (a)
print (b)
print (c)
```

This produces the following result:

```
1
2
Zara Ali
```

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

Python Variable Names

Every Python variable should have a unique name like a, b, c. A variable name can be meaningful like color, age, name etc. There are certain rules which should be taken care while naming a Python variable:

- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number or any special character like $, (, * % etc.
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Python variable names are case-sensitive which means Name and NAME are two different variables in Python.
- Python reserved keywords cannot be used naming the variable.

Example

Following are valid Python variable names:

```
counter = 100
_count  = 100
name1 = “Zara”
name2 = “Nuha”
Age  = 20
zara_salary = 100000

print (counter)
print (_count)
print (name1)
print (name2)
print (Age)
print (zara_salary)
```

This will produce the following result:

```
100
100
Zara
Nuha
20
100000
```

Example

Following are invalid Python variable names:

```
1counter = 100
$_count = 100
zara-salary = 100000
print (1counter)
print ($count)
print (zara-salary)
```

This will produce the following result:

```
File “main.py”, line 3
 1counter = 100
 ^
SyntaxError: invalid syntax
```

Python Local Variable

Python Local Variables are defined inside a function. We can not access variable outside the function.

> A Python functions is a piece of reusable code and you will learn more about function in Python - Functions tutorial.

Following is an example to show the usage of local variables:

```
def sum(x,y):
   sum = x + y
   return sum
print(sum(5, 10))
```

```
15
```

Python Global Variable

Any variable created outside a function can be accessed within any function and so they have global scope. Following is an example of global variables:

```
x = 5
y = 10
def sum():
   sum = x + y
   return sum
print(sum())
```

This will produce the following result:

```
15
```

Python - Data Types

Python Data Types are used to define the type of a variable. It defines what type of data we are going to store in a variable. 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 built-in data types which we will discuss with in this tutorial:

- Numeric - int, float, complex
- String - str
- Sequence - list, tuple, range
- Binary - bytes, bytearray, memoryview
- Mapping - dict
- Boolean - bool
- Set - set, frozenset
- None - NoneType

Python Numeric Data Type

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

```
var1 = 1
var2 = 10
var3 = 10.023
```

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.

Example

Following is an example to show the usage of Integer, Float and Complex numbers:

```
# integer variable.
a=100
print("The type of variable having value", a, " is ", type(a))

# float variable.
b=20.345
print("The type of variable having value", b, " is ", type(b))

# complex variable.
c=10+3j
print("The type of variable having value", c, " is ", type(c))
```

Python String Data Type

Python Strings 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 in Python. For example −

```
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 List Data Type

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

The values stored in a Python 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 −

```
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 Tuple Data Type

Python tuple is another sequence data type that is similar to a list. A Python 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 −

```
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 −

```
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
```

Python Ranges

Python **range**() is an in-built function in Python which returns a sequence of numbers starting from 0 and increments to 1 until it reaches a specified number.

We use **range**() function with for and while loop to generate a sequence of numbers. Following is the syntax of the function:

```
range(start, stop, step)
```

Here is the description of the parameters used:

- **start**: Integer number to specify starting position, (Its optional, Default: 0)
- **stop**: Integer number to specify starting position (It's mandatory)
- **step**: Integer number to specify increment, (Its optional, Default: 1)

Examples

Following is a program which uses for loop to print number from 0 to 4 −

```
for i in range(5):
  print(i)
```

This produce the following result −

```
0
1
2
3
4
```

Now let's modify above program to print the number starting from 1 instead of 0:

```
for i in range(1, 5):
  print(i)
```

This produce the following result −

```
1
2
3
4
```

Again, let's modify the program to print the number starting from 1 but with an increment of 2 instead of 1:

```
for i in range(1, 5, 2):
  print(i)
```

This produce the following result −

```
1
3
```

Python Dictionary

Python dictionaries are kind of hash table type. They work like associative arrays or hashes found in Perl and consist of key-value pairs. A dictionary key can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.

Dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessed using square braces ([]). For example −

```
dict = {}
dict[‘one’] = “This is one”
dict[2]     = “This is two”

tinydict = {‘name’: ‘john’,’code’:6734, ‘dept’: ‘sales’}


print (dict[‘one’])       # Prints value for ‘one’ key
print (dict[2])           # Prints value for 2 key
print (tinydict)          # Prints complete dictionary
print (tinydict.keys())   # Prints all the keys
print (tinydict.values()) # Prints all the values
```

This produce the following result −

```
This is one
This is two
{‘dept’: ‘sales’, ‘code’: 6734, ‘name’: ‘john’}
[‘dept’, ‘code’, ‘name’]
[‘sales’, 6734, ‘john’]
```

Python dictionaries have no concept of order among elements. It is incorrect to say that the elements are “out of order”; they are simply unordered.

Python Boolean Data Types

Python **boolean** type is one of built-in data types which represents one of the two values either **True** or **False**. Python **bool**() function allows you to evaluate the value of any expression and returns either True or False based on the expression.

Examples

Following is a program which prints the value of boolean variables a and b −

```
a = True
# display the value of a
print(a)

# display the data type of a
print(type(a))
```

This produce the following result −

```
true
<class 'bool'>
```

Following is another program which evaluates the expressions and prints the return values:

```
# Returns false as a is not equal to b
a = 2
b = 4
print(bool(a==b))

# Following also prints the same
print(a==b)

# Returns False as a is None
a = None
print(bool(a))

# Returns false as a is an empty sequence
a = ()
print(bool(a))

# Returns false as a is 0
a = 0.0
print(bool(a))

# Returns false as a is 10
a = 10
print(bool(a))
```

This produce the following result −

```
False
False
False
False
False
True
```

Python Data Type Conversion

Sometimes, you may need to perform conversions between the built-in data types. To convert data between different Python data types, you simply use the type name as a function.

Conversion to int

Following is an example to convert number, float and string into integer data type:

```
a = int(1)     # a will be 1
b = int(2.2)   # b will be 2
c = int("3")   # c will be 3

print (a)
print (b)
print (c)
```

This produce the following result −

```
1
2
3
print (c)
```

Conversion to float

Following is an example to convert number, float and string into float data type:

```
a = float(1)     # a will be 1.0
b = float(2.2)   # b will be 2.2
c = float("3.3") # c will be 3.3

print (a)
print (b)
print (c)
```

This produce the following result −

```
1.0
2.2
3.3
```

Conversion to string

Following is an example to convert number, float and string into string data type:

```
a = str(1)     # a will be "1"
b = str(2.2)   # b will be "2.2"
```

```
c = str("3.3") # c will be "3.3"

print (a)
print (b)
print (c)
```

This produce the following result −

```
1
2.2
3.3
```

Data Type Conversion Functions

There are several built-in functions to perform conversion from one data type to another. These functions return a new object representing the converted value.

Sr.No.	Function & Description
1	**int(x [,base])** Converts x to an integer. base specifies the base if x is a string.
2	**long(x [,base])** Converts x to a long integer. base specifies the base if x is a string.
3	**float(x)** Converts x to a floating-point number.
4	**complex(real [,imag])** Creates a complex number.
5	**str(x)** Converts object x to a string representation.
6	**repr(x)** Converts object x to an expression string.

7	**eval(str)** Evaluates a string and returns an object.
8	**tuple(s)** Converts s to a tuple.
9	**list(s)** Converts s to a list.
10	**set(s)** Converts s to a set.
11	**dict(d)** Creates a dictionary. d must be a sequence of (key,value) tuples.
12	**frozenset(s)** Converts s to a frozen set.
13	**chr(x)** Converts an integer to a character.
14	**unichr(x)** Converts an integer to a Unicode character.
15	**ord(x)** Converts a single character to its integer value.
16	**hex(x)** Converts an integer to a hexadecimal string.
17	**oct(x)** Converts an integer to an octal string.

Python - Operators

Python operators are the constructs which can manipulate the value of operands. These are symbols used for the purpose of logical, arithmetic and various other operations.

Consider the expression 4 + 5 = 9. Here, 4 and 5 are called operands and + is called operator. In this tutorial, we will study different types of Python operators.

Types of Python Operators

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 quick look on all these operators one by one.

Python Arithmetic Operators

Python arithmetic operators are used to perform mathematical operations on numerical values. These operations are Addition, Subtraction, Multiplication, Division, Modulus, Expoents and Floor Division.

Operator	Name	Example
+	Addition	10 + 20 = 30
-	Subtraction	20 – 10 = 10
*	Multiplication	10 * 20 = 200
/	Division	20 / 10 = 2
%	Modulus	22 % 10 = 2
**	Exponent	4**2 = 16
//	Floor Division	9//2 = 4

Example

Following is an example which shows all the above operations:

```
a = 21
b = 10

# Addition
print (“a + b : “, a + b)

# Subtraction
print (“a - b : “, a - b)

# Multiplication
print (“a * b : “, a * b)

# Division
print (“a / b : “, a / b)

# Modulus
print (“a % b : “, a % b)

# Exponent
print (“a ** b : “, a ** b)

# Floor Division
print (“a // b : “, a // b)
```

This produce the following result −

```
a + b :  31
a - b :  11
a * b :  210
a / b :  2.1
a % b :  1
a ** b :  16679880978201
a // b :  2
```

Python Comparison Operators

Python comparison operators compare the values on either sides of them and decide the relation among them. They are also called relational operators. These operators are equal, not equal, greater than, less than, greater than or equal to and less than or equal to.

Operator	Name	Example
==	Equal	4 == 5 is not true.

!=	Not Equal	4 != 5 is true.
>	Greater Than	4 > 5 is not true.
<	Less Than	4 < 5 is true.
>=	Greater than or Equal to 4	>= 5 is not true.
<=	Less than or Equal to	4 <= 5 is true.

Example

Following is an example which shows all the above comparison operations:

```
a = 4
b = 5

# Equal
print (“a == b : “, a == b)

# Not Equal
print (“a != b :
 “, a != b)

# Greater Than
print (“a > b : “, a > b)

# Less Than
print (“a < b : “, a < b)

# Greater Than or Equal to
print (“a >= b : “, a >= b)

# Less Than or Equal to
print (“a <= b : “, a <= b)
```

This produce the following result −

```
a == b :  False
a != b :  True
a > b :  False
a < b :  True
a >= b :  False
a <= b :  True
```

Python Assignment Operators

Python assignment operators are used to assign values to variables. These operators include simple assignment operator, addition assign, subtraction assign, multiplication assign, division and assign operators etc.

Operator	Name	Example
=	Assignment Operatora	= 10
+=	Addition Assignment	a += 5 (Same as a = a + 5)
-=	Subtraction Assignment	a -= 5 (Same as a = a - 5)
*=	Multiplication Assignment	a *= 5 (Same as a = a * 5)
/=	Division Assignment	a /= 5 (Same as a = a / 5)
/=	Division Assignment	a /= 5 (Same as a = a / 5)
**=	Exponent Assignment	a **= 2 (Same as a = a ** 2)
//=	Floor Division Assignment	a //= 3 (Same as a = a // 3)

Example

Following is an example which shows all the above assignment operations:

```
# Assignment Operator
a = 10

# Addition Assignment
a += 5
print (“a += 5 : “, a)

# Subtraction Assignment
a -= 5
print (“a -= 5 : “, a)

# Multiplication Assignment
a *= 5
print (“a *= 5 : “, a)
```

```
# Division Assignment
a /= 5
print (“a /= 5 : “,a)

# Remainder Assignment
a %= 3
print (“a %= 3 : “, a)

# Exponent Assignment
a **= 2
print (“a **= 2 : “, a)

# Floor Division Assignment
a //= 3
print (“a //= 3 : “, a)
```

This produce the following result −

```
a += 5 :  105
a -= 5 :  100
a *= 5 :  500
a /= 5 :  100.0
a %= 3 :  1.0
a **= 2 :  1.0
a //= 3 :  0.0
```

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 = 12 (0000 1100)

a|b = 61 (0011 1101)

a^b = 49 (0011 0001)

~a = -61 (1100 0011)

a << 2 = 240 (1111 0000)

a>>2 = 15 (0000 1111)

There are following Bitwise operators supported by Python language

Operator	Name	Example
&	Binary AND	Sets each bit to 1 if both bits are 1
|	Binary OR	Sets each bit to 1 if one of two bits is 1
^	Binary XOR	Sets each bit to 1 if only one of two bits is 1
~	Binary Ones Complement	Inverts all the bits
<<	Binary Left Shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off
>>	Binary Right Shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off

Example

Following is an example which shows all the above bitwise operations:

```
a = 60          # 60 = 0011 1100
b = 13          # 13 = 0000 1101

# Binary AND
c = a & b       # 12 = 0000 1100
print (“a & b : “, c)

# Binary OR
c = a | b       # 61 = 0011 1101
print (“a | b : “, c)

# Binary XOR
c = a ^ b       # 49 = 0011 0001
print (“a ^ b : “, c)

# Binary Ones Complement
c = ~a;         # -61 = 1100 0011
print (“~a : “, c)

# Binary Left Shift
c = a << 2;     # 240 = 1111 0000
print (“a << 2 : “, c)
```

```
# Binary Right Shift
c = a >> 2;       # 15 = 0000 1111
print (“a >> 2 : “, c)
```

This produce the following result −

```
a & b :  12
a | b :  61
a ^ b :  49
~a :  -61
a >> 2 :  240
a >> 2 :  15
```

Python Logical Operators

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

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	sed 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 −

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.

Example

```
#!/usr/bin/python

a = 10
b = 20
list = [1, 2, 3, 4, 5 ];

if ( a in list ):
   print “Line 1 - a is available in the given list”
else:
   print “Line 1 - a is not available in the given list”

if ( b not in list ):
   print “Line 2 - b is not available in the given list”
else:
   print “Line 2 - b is available in the given list”

a = 2
if ( a in list ):
   print “Line 3 - a is available in the given list”
else:
   print “Line 3 - a is not available in the given list”
```

When you execute the above program it produces the following result −

```
Line 1 - a is not available in the given list
Line 2 - b is not available in the given list
Line 3 - a is available in the given list
```

Python Identity Operators

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

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).

Example

```
#!/usr/bin/python

a = 20
b = 20

if ( a is b ):
   print “Line 1 - a and b have same identity”
else:
   print “Line 1 - a and b do not have same identity”

if ( id(a) == id(b) ):
   print “Line 2 - a and b have same identity”
else:
   print “Line 2 - a and b do not have same identity”

b = 30
if ( a is b ):
   print “Line 3 - a and b have same identity”
else:
   print “Line 3 - a and b do not have same identity”

if ( a is not b ):
   print “Line 4 - a and b do not have same identity”
else:
   print “Line 4 - a and b have same identity”
```

When you execute the above program it produces the following result −

```
Line 1 - a and b have same identity
Line 2 - a and b have same identity
Line 3 - a and b do not have same identity
Line 4 - a and b do not have same identity
```

Python Operators Precedence

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

Operator	Description
**	Exponentiation (raise to the power)
~ + -	Complement, unary plus and minus (method names for the last two are +@ and -@)
* / % //	Multiply, divide, modulo and floor division
>> <<	Right and left bitwise shift
&	Bitwise ‘AND’td>

^ |	Bitwise exclusive `OR' and regular `OR'
<= < > >=	Comparison operators
<> == !=	Equality operators
= %= /= //= -= += *= **=	Assignment operators
is is not	Identity operators
in not in	Membership operators
not or and	Logical operators

Operator precedence affects how an expression is evaluated.

For example, x = 7 + 3 * 2; here, x is assigned 13, not 20 because operator * has higher precedence than +, so it first multiplies 3*2 and then adds into 7.

Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom.

Example

```
#!/usr/bin/python

a = 20
b = 10
c = 15
d = 5
e = 0

e = (a + b) * c / d       #( 30 * 15 ) / 5
print "Value of (a + b) * c / d is ",  e

e = ((a + b) * c) / d     # (30 * 15 ) / 5
print "Value of ((a + b) * c) / d is ",  e

e = (a + b) * (c / d);    # (30) * (15/5)
print "Value of (a + b) * (c / d) is ",  e

e = a + (b * c) / d;      #  20 + (150/5)
print "Value of a + (b * c) / d is ",  e
```

When you execute the above program, it produces the following result −

```
Value of (a + b) * c / d is 90
Value of ((a + b) * c) / d is 90
Value of (a + b) * (c / d) is 90
Value of a + (b * c) / d is 50
```

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

Next Steps?

Have any questions while reading the eBook? Want to discuss programming, gadgets and gaming? You can email us (kaustav@gizmofacts.com) where you can ask any questions you have in min2d and subscribe to our Newsletter and visit our Facebook Page.

You can also check our blog "Gizmofacts" where you can find a ton of useful "how-to-guides", tutorials around blogging, programming, gadget reviews and gaming.