**Unit 1**

**Introduction to Python**

**Introduction:-**

Python is developed by **Guido van Rossum**. Guido van Rossum started implementing Python in 1989. Python is a very simple programming language so even if you are new to programming, you can learn python without facing any issues.

**What can Python do?**

* Python can be used on a server to create web applications.
* Python can be used alongside software to create workflows.
* Python can connect to database systems. It can also read and modify files.
* Python can be used to handle big data and perform complex mathematics.
* Python can be used for rapid prototyping, or for production-ready software development.

**Why Python?**

* Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
* Python has a simple syntax similar to the English language.
* Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
* Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
* Python can be treated in a procedural way, an object-orientated way or a functional way.

**Why the Name Python?**

* There is a fact behind choosing the name [Python](https://www.javatpoint.com/python-tutorial). **Guido van Rossum** was reading the script of a popular BBC comedy series "**Monty Python's Flying Circus**". It was late on-air 1970s.
* Van Rossum wanted to select a name which unique, sort, and little-bit mysterious. So he decided to select naming Python after the **"Monty Python's Flying Circus"** for their newly created programming language.
* The comedy series was creative and well random. It talks about everything. Thus it is slow and unpredictable, which made it very interesting.
* Python is also versatile and widely used in every technical field, such as [Machine Learning](https://www.javatpoint.com/machine-learning), [Artificial Intelligence](https://www.javatpoint.com/artificial-intelligence-tutorial), Web Development, [Mobile Application](https://www.javatpoint.com/javatpoint.com/mobile-application-testing), Desktop Application, Scientific Calculation, etc.

## 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.

### Macintosh Installation

Recent Macs come with Python installed, but it may be several years out of date. See[http://www.python.org/download/mac/](https://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−

**In the csh shell** − type setenv PATH "$PATH:/usr/local/bin/python" and press Enter.

**In the bash shell (Linux)** − type export ATH="$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.

## Python Environment Variables:-

|  |  |
| --- | --- |
| **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. |

### Different ways to run Python Script

* Here are the ways with which we can run a Python script.
* Interactive Mode
* Command Line
* Text Editor (VS Code)
* IDE (PyCharm**)**

### 1)Interactive Interpreter

In Interactive Mode, you can run your script line by line in a sequence.

To enter in an interactive mode, you will have to open Command Prompt on your windows machine and type ‘python’ and press Enter

Run the following lines one by one in the interactive mode:

|  |
| --- |
| name = "Aakash"  print("My name is " + name) |

### 

### 2)Command-line

To run a Python script store in a ‘.py’ file in command line, we have to write ‘python’ keyword before the file name in the command prompt.

python hello.py

You can write your own file name in place of ‘hello.py’.

**Output:**

****

**3) Text Editor (VS Code)**  
To run Python script on a text editor like [VS Code (Visual Studio Code)](https://code.visualstudio.com/) then you will have to do the following:

* Go in the extension section or press ‘Ctrl+Shift+X’ on windows, then search and install the extension named ‘Python’ and ‘Code Runner’. Restart your vs code after that.
* Now, create a new file with the name ‘**hello.py**’ and write the below code in it:
* print('Hello World!')
* Then, right click anywhere in the text area and select the option that says ‘Run Code’ or press ‘Ctrl+Alt+N’ to run the code.

**Output:**



**4) IDE (PyCharm)**  
To run Python script on a [IDE (Integrated Development Environment) like PyCharm](https://www.jetbrains.com/pycharm/), you will have to do the following:

* Create a new project.
* Give a name to that project as ‘GfG’ and click on Create.
* Select the root directory with the project name we specified in the last step. **Right click** on it, go in **New** and click on ‘**Python file**’ option. Then give the name of the file as ‘**hello**’ (you can specify any name as per your project requirement). This will create a ‘**hello.py**’ file in the project root directory.  
  **Note:** You don’t have to specify the extension as it will take it automatically.



# [Running Python script from IDLE on Windows 7 64 bit](https://stackoverflow.com/questions/6513967/running-python-script-from-idle-on-windows-7-64-bit)

1. Run IDLE. You will be presented with the "Python Shell" window and a >>> prompt.
2. Click File, New Window. You will be presented with an "Untitled" window for editing a script.
3. Enter your script in the "Untitled" window.
4. In the "Untitled" window, select Run, Run Module (or press F5) to run your script.
5. A dialog "Source Must Be Saved. OK to Save?" appears. Click OK.
6. In the Save As dialog:  
   a. Browse to a directory to save your script.  
   b. Enter a filename.  
   c. Click Save.
7. The "Python Shell" window will display the output of your script.
8. Edit script and press F5 as needed to re-run your script.

**Features / Characteristics of python:-**

There are many features in Python, some of which are discussed below –

**1. Easy to code:**

Python is a high-level programming language. Python is very easy to learn the language as compared to other languages like C, C#, Javascript, Java, etc. It is very easy to code in python language and anybody can learn python basics in a few hours or days. It is also a developer-friendly language.

**2. Free and Open Source:**

Python language is freely available at the official website and you can download it from the given download link below click on the Download Python keyword.

Since it is open-source, this means that source code is also available to the public. So you can download it as, use it as well as share it.

**3. Object-Oriented Language:**

One of the key features of python is Object-Oriented programming. Python supports object-oriented language and concepts of classes, objects encapsulation, etc.

**4. GUI Programming Support:**

Graphical User interfaces can be made using a module such as PyQt5, PyQt4, wxPython, or Tk in python.

PyQt5 is the most popular option for creating graphical apps with Python.

**5. High-Level Language:**

Python is a high-level language. When we write programs in python, we do not need to remember the system architecture, nor do we need to manage the memory.

**6. Extensible feature:**

Python is a Extensible language. We can write us some Python code into C or C++ language and also we can compile that code in C/C++ language.

**7. Python is Portable language:**

Python language is also a portable language. For example, if we have python code for windows and if we want to run this code on other platforms such as Linux, Unix, and Mac then we do not need to change it, we can run this code on any platform.

**8. Python is Integrated language:**

Python is also an Integrated language because we can easily integrated python with other languages like c, c++, etc.

**9. Interpreted Language:**

Python is an Interpreted Language because Python code is executed line by line at a time. like other languages C, C++, Java, etc. there is no need to compile python code this makes it easier to debug our code. The source code of python is converted into an immediate form called bytecode.

**10. Large Standard Library**

Python has a large standard library which provides a rich set of module and functions so you do not have to write your own code for every single thing. There are many libraries present in python for such as regular expressions, unit-testing, web browsers, etc.

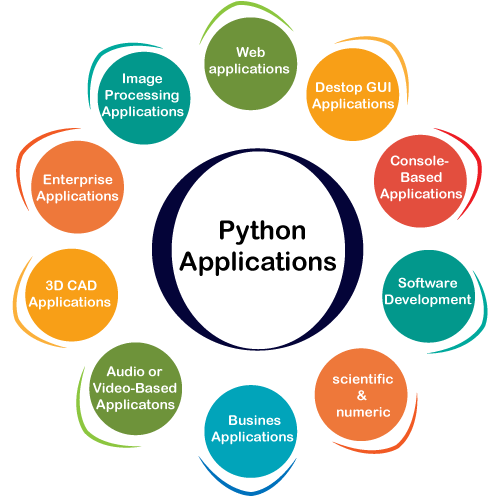
**11. Dynamically Typed Language:**

Python is a dynamically-typed language. That means the type (for example- int, double, long, etc.) for a variable is decided at run time not in advance because of this feature we don’t need to specify the type of variable.

**Applications of Python:-**

## Python is known for its general-purpose nature that makes it applicable in almost every domain of software development. Python makes its presence in every emerging field. It is the fastest-growing programming language and can develop any application.

Here, we are specifying application areas where Python can be applied.



### 1) Web Applications

We can use Python to develop web applications. It provides libraries to handle internet protocols such as HTML and XML, JSON, Email processing, request, beautifulSoup, Feedparser, etc. One of Python web-framework named Django is used on **Instagram**. Python provides many useful frameworks, and these are given below:

* Django and Pyramid framework(Use for heavy applications)
* Flask and Bottle (Micro-framework)
* Plone and Django CMS (Advance Content management)

### 2) Desktop GUI Applications

The GUI stands for the Graphical User Interface, which provides a smooth interaction to any application. Python provides a **Tk GUI library** to develop a user interface. Some popular GUI libraries are given below.

* Tkinter or Tk
* wxWidgetM
* Kivy (used for writing multitouch applications )
* PyQt or Pyside

### 3) Console-based Application

Console-based applications run from the command-line or shell. These applications are computer program which are used commands to execute. This kind of application was more popular in the old generation of computers. Python can develop this kind of application very effectively. It is famous for having REPL, which means **the Read-Eval-Print Loop** that makes it the most suitable language for the command-line applications.

Python provides many free library or module which helps to build the command-line apps. The necessary **IO** libraries are used to read and write. It helps to parse argument and create console help text out-of-the-box. There are also advance libraries that can develop independent console apps.

### 4) Software Development

Python is useful for the software development process. It works as a support language and can be used to build control and management, testing, etc.

* **SCons** is used to build control.
* **Buildbot** and **Apache** Gumps are used for automated continuous compilation and testing.
* **Round** or **Trac** for bug tracking and project management.

### 5) Scientific and Numeric

This is the era of Artificial intelligence where the machine can perform the task the same as the human. Python language is the most suitable language for Artificial intelligence or machine learning. It consists of many scientific and mathematical libraries, which makes easy to solve complex calculations.

Implementing machine learning algorithms require complex mathematical calculation. Python has many libraries for scientific and numeric such as Numpy, Pandas, Scipy, Scikit-learn, etc. If you have some basic knowledge of Python, you need to import libraries on the top of the code. Few popular frameworks of machine libraries are given below.

* SciPy
* Scikit-learn
* NumPy
* Pandas
* Matplotlib

### 6) Business Applications

Business Applications differ from standard applications. E-commerce and ERP are an example of a business application. This kind of application requires extensively, scalability and readability, and Python provides all thesefeatures.Oddo is an example of the all-in-one Python-based application which offers a range of business applications. Python provides a **Tryton** platform which is used to develop the business application.

### 7) Audio or Video-based Applications

Python is flexible to perform multiple tasks and can be used to create multimedia applications. Some multimedia applications which are made by using Python are **TimPlayer, cplay,** etc. The few multimedia libraries are given below.

* Gstreamer
* Pyglet
* QT Phonon

### 8) 3D CAD Applications

The CAD (Computer-aided design) is used to design engineering related architecture. It is used to develop the 3D representation of a part of a system. Python can create a 3D CAD application by using the following functionalities.

* Fandango (Popular )
* CAMVOX
* HeeksCNC
* AnyCAD
* RCAM

### 9) Enterprise Applications

Python can be used to create applications that can be used within an Enterprise or an Organization. Some real-time applications are OpenERP, Tryton, Picalo, etc.

### 10) Image Processing Application

Python contains many libraries that are used to work with the image. The image can be manipulated according to our requirements. Some libraries of image processing are given below.

* OpenCV
* Pillow
* SimpleITK

## Comments in Python

Comments are ignore by the Python interpreter, therefore comments helps us in debugging the program. Comments in python, also helps in reviewing the python code later. All characters after hash (#) sign referred as comment, up to the physical line end. Here is an example of comment in Python:

# Python Basic Syntax

**Example**

# We are comments

print("Hello World, I am Python"); # i am also comment

**Python Virtual Machine:-**

The main data structure in the PVM is the "regular" stack (which is like a

restricted a list push=append, pop=pop). A stack's primary operations are

load/push and store/pop. We load/push a value on the top of an upwardly-growing

stack (incrementing the stackp -stack pointer- that indexes the top); we

store/pop a values from the top of a stack (decrementing the stackp).

There is a secondarily important block stack that is used to store information about nested loops, try, and with statements. For example, a break statement is transated into code that uses the block stack to determine which loop to

break out of (and how to continue executing at the first statement outside the

loop). As loops, try/except, and with statements are started, information about

their blocks are pushed onto the block stack; as they terminate, this

information is popped off the bock stack. The block stack is too complicated

for today's lecture and is not needed to understand it: so when we run across

block stack instructions we will point out the fact that we are ignoring them.

Here is an example of a simple sequence of stack operations to perform the

calculation d = a+b\*c, assuming that a, b, c, and d are local variables inside

a function: assume co\_varnames is ('a', 'b', 'c', 'd') and the actual values

for these names are stored in a parallel tuple (1, 2, 3, None): e.g., the value

for 'a' is 1, the value for 'b' is 2, the value for 'c' is 3, and the value for

'd' is None. Generally the value for a name at index i in the co\_varnames tuple

is stored in index i in the tuple of actual values.

**Python Keywords**

Keywords are the reserved words in Python.

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

In Python, keywords are case sensitive.There are 33 keywords in Python 3.All the keywords except True, False and None are in lowercase and they must be written as it is. The list of all the keywords are given below.

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

**Python Identifiers**

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

**Rules for writing identifiers**

1. Identifiers can be a combination of letters in lowercase (a to z) or uppercase (A to Z) or digits (0 to 9) or an underscore (\_). Names like myClass, var\_1 and print\_this\_to\_screen, all are valid example.
2. An identifier cannot start with a digit. 1variable is invalid, butvariable1 is perfectly fine.
3. Keywords cannot be used as identifiers.

>>>global = 1

File "<interactive input>", line 1

global = 1

^

SyntaxError: invalid syntax

1. We cannot use special symbols like !, @, #, $, % etc. in our identifier.

>>>a@ = 0

File "<interactive input>", line 1

a@ = 0

^

SyntaxError: invalid syntax

1. Identifier can be of any length.

**Operators:-**

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**

* Arithmetic Operators
* Comparison (Relational) Operators
* AssignmentOperators
* Logical Or Boolean Operators
* Bitwise Operators
* Membership Operators
* Identity Operators
  1. **Python Arithmetic Operators**

Arithmetic operators are used to perform arithmetic operations between two operands. It includes + (addition), - (subtraction), \*(multiplication), /(divide), %(reminder), //(floor division), and exponent (\*\*) operators.

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

|  |  |  |
| --- | --- | --- |
| 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 |

**Example:-**

x = 10

y = 20

print('x + y =',x+y)

print('x - y =',x-y)

print('x \* y =',x\*y)

print('x / y =',x/y)

print('x // y =',x//y)

print('x \*\* y =',x\*\*y)

**2) Python Comparison Operators**

Comparison operators are used to comparing the value of the two operands and returns Boolean true or false accordingly. The comparison operators are described in the following table..

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

|  |  |  |
| --- | --- | --- |
| **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. |

**Example:-**

x = 10

y = 20

print('x > y is',x>y)

print('x < y is',x<y)

print('x == y is',x==y)

print('x != y is',x!=y)

print('x >= y is',x>=y)

print('x <= y is',x<=y)

**3) Python Assignment Operators**

The assignment operators are used to assign the value of the right expression to the left operand. The assignment operators are described in the following table.

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

|  |  |  |
| --- | --- | --- |
| **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 / ac /= 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 |

**Example:-**

a = 10

b = 20

c = 0

c = a + b

print "Line 1 - Value of c is ", c

c += a

print "Line 2 - Value of c is ", c

c \*= a

print "Line 3 - Value of c is ", c

c /= a

print "Line 4 - Value of c is ", c

c = 2

c %= a

print "Line 5 - Value of c is ", c

c \*\*= a

print "Line 6 - Value of c is ", c

c //= a

print "Line 7 - Value of c is ", c

**4)Python Bitwise Operators**

The bitwise operators perform bit by bit operation on the values of the two operands. Assume if a = 60; and b = 13; Now in binary format they will be as follows −

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

|  |  |  |
| --- | --- | --- |
| **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) |

**Example:-**

a = 60 # 60 = 0011 1100

b = 13 # 13 = 0000 1101

c = 0

c = a & b; # 12 = 0000 1100

print "Line 1 - Value of c is ", c

c = a | b; # 61 = 0011 1101

print "Line 2 - Value of c is ", c

c = a ^ b; # 49 = 0011 0001

print "Line 3 - Value of c is ", c

c = ~a; # -61 = 1100 0011

print "Line 4 - Value of c is ", c

c = a << 2; # 240 = 1111 0000

print "Line 5 - Value of c is ", c

c = a >> 2; # 15 = 0000 1111

print "Line 6 - Value of c is ", c

**5) Python Logical Operators**

The logical operators are used primarily in the expression evaluation to make a decision. Python supports the following logical operators

|  |  |
| --- | --- |
| **Operator** | **Description** |
| and | If both the expression are true, then the condition will be true. If a and b are the two expressions, a → true, b → true => a and b → true. |
| or | If one of the expressions is true, then the condition will be true. If a and b are the two expressions, a → true, b → false => a or b → true. |
| not | If an expression **a** is true, then not (a) will be false and vice versa. |

**Example:-**

x = True

y = False

print('x and y is',x and y)

print('x or y is',x or y)

print('not x is',not x)

**6)Python Membership Operators**

Python membership operators are used to check the membership of value inside a Python data structure. If the value is present in the data structure, then the resulting value is true otherwise it returns false.

|  |  |  |
| --- | --- | --- |
| **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:-**

x = 'Hello world'

y = {1:'a',2:'b'}

print('H' in x)

print('hello' not in x)

print(1 in y)

print('a' in y)

**7)Python Identity Operators**

The identity operators are used to decide whether an element certain class or type.

|  |  |  |
| --- | --- | --- |
| **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:-**

x1 = 5

y1 = 5

x2 = 'Hello'

y2 = 'Hello'

x3 = [1,2,3]

y3 = [1,2,3]

print(x1 is not y1)

print(x2 is y2)

print(x3 is y3)

**Precedence Operator:-**

Python has well-defined rules for specifying the order in which the operators in an expression are evaluated when the expression has several operators. For example, multiplication and division have a higher precedence than addition and subtraction. Precedence rules can be overridden by explicit parentheses.

**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 |
| + - | Addition and subtraction |
| >><< | 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**

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

**Output:-**

Value of (a + b)\* c / d is90

Value of ((a + b)\* c)/ d is90

Value of (a + b)\*(c / d)is90

Value of a +(b \* c)/ d is50

**Python Operator Associativity**

In the above table, you can confirm that some of the groups have many operators. It means that all operators in a group are at the same precedence level.

And whenever two or more operators have the same precedence, then associativity defines the order of operations.

Hence, associativity is the order in which Python evaluates an expression containing multiple operators of the same precedence. Almost all operators except the exponent (\*\*) support the left-to-right associativity.

**For example**

the product (\*) and the modulus (%) have the same precedence. So, if both appear in an expression, then the left one will get evaluated first.

# Testing Left-right associativity

# Result: 1

print(4 \* 7 % 3)

# Testing left-right associativity

# Result: 0

print(2 \* (10 % 5))

As said earlier, the only operator which has right-to-left associativity in Python is the exponent (\*\*) operator.

**Example:-**

# Checking right-left associativity of \*\* exponent operator

# Output: 256

print(4 \*\* 2 \*\* 2)

# Checking the right-left associativity

# of \*\*

# Output: 256

print((4 \*\* 2) \*\* 2)

You might have observed that the ‘print(4 \*\* 2 \*\* 2)’ is similar to ‘(4 \*\* 2 \*\* 2).