PYTHON PROGRAMMING LANGUAGE

Introduction

Python is a high-level, interpreted, interactive and object-oriented scripting language. Designed to be highly readable. It uses English keywords frequently so that we can easily learn and use this language. It mixes good features from different languages like java and perl. These days, from data to web development, Python has emerged as a very powerful and popular language.

Python Users

- The YouTube video sharing service is largely written in Python.
- IRobot uses Python to develop commercial robotic vacuum cleaners.

History of Python

Python is a fairly old language created by Guido Van Rossum. The design began in the late 1980s and was first released in February 1991 and the current version is 3.6.5.

Why Python was created?

In late 1980s, Guido Van Rossum was working on the Amoeba distributed operating system group. He wanted to use an interpreted language like ABC that could access the Amoeba system calls. So, he decided to create a language that was extensible. This led to a design of new language which was later named Python.

Why the name Python?

It wasn't named after a dangerous snake. Rossum was fan of a comedy series from late seventies. The name "Python" was adopted from the same series "Monty Python's Flying Circus".

Features of Python Programming

1. **Easy to learn**
   A python program is clearly defined and easily readable. The structure of the program is very simple. Use few keywords and a clearly defined syntax. This makes it easy for just anyone to pick up the language quickly.
2. **Simple:** Python is a simple and small language. Reading a program written in python feels almost like reading English. This is in fact the greatest strength of python which allows programmes to concentrate on the solution to the problem rather than the language itself.
3. **Versatile**
   Python supports development of a wide range of applications ranging from simple test processing to WWW browsers to games.
4. **Free and Open Source**
   Python is an example of open source software. Therefore, anyone can freely distribute it, read the source code, edit it, and even use the code to write new programs.
5. **High-level Language**
   When writing in python, the programmers don't have to worry about the low-level details like managing memory used by the program, etc. They just need to concentrate on the writing solution of the current problem at hand.
6. **Interactive**
   Programs in python work in interactive mode which allows interactive testing and debugging of pieces of code. Programmers can easily interact with the interpreter directly at the python prompt to write their programs.
7. **Portable**

Python is portable language and hence the programs behave the same on a wide variety of hardware platforms and have the same interface on all platforms. The programs work on any of the operating system like Linux, windows etc. without requiring any changes.

8. **Interpreted**
   Python is processed at run-time by the interpreter. So there is no need to compile a program before executing it. You can simply run the program. Basically Python converts the source code into an intermediate form called Bytecode, which is then translated into the native language of your computer so that it can be executed. Bytecode make the python code portable since user just have to copy the code and run it without worrying about compiling, linking, and loading process.
9. **Dynamic**
   Python executes dynamically. Programs written in python can be copied and used for flexible development of applications. If there is any error, it is reported at run-time to allow interactive program development.
10. **Extensible**
    Since python is an open source software, anyone can add low-level modules to the python interpreter. These modules enable programmers to add to or customize their tools to work more efficiently .
11. **Embeddable**
    Programmers can embed python within their C, C++,COM, ActiveX, CORBA and Java programs to give 'scripting' capabilities for users.
12. **Extensive Libraries**
    Python has a huge library that is easily portable across different platforms. These library functions are compatible on UNIX, Windows etc. And allow programmers to perform wide range of applications varying from text processing, maintaining, database, to GUI programming.

Applications of Python

1. **Embedded scripting language:**
   Python is used as an embedded scripting language for various testing/ building/ deployment/ monitoring frameworks, scientific apps, and quick scripts.
2. **3D Software:**
   3D software like Maya uses Python for automating small user tasks, or for doing more complex integration such as talking to databases and asset management systems.
3. **Web development:**
   Python is an easily extensible language that provides good integration with database and other web standards.
4. **GUI-based desktop applications:**
   Simple syntax, modular architecture, rich text processing tools and the ability to work on multiple operating systems makes Python a preferred choice for developing desktop-based applications.
5. **Image processing and graphic design applications:**

Python is used to make 2D imaging software such as Inkscape, GIMP, Paint Shop Pro and Scribus. It is also used to make 3D animation packages, like Blender, 3ds Max, Cinema 4D, Houdini, Lightwave and Maya.

6. **Scientific and computational applications:**
   Features like high speed, productivity and availability of tools, such as Scientific Python and Numeric Python, have made Python a preferred language to perform computation and processing of scientific data. 3D modeling software, such as FreeCAD, and finite element method software, like Abaqus, are coded in Python.
7. **Games:**
   Python has various modules, libraries, and platforms that support development of games. Games like Civilization-IV, Disney's Toontown Online, Vega Strike, etc. are coded using Python.
8. **Enterprise and business applications:**
   Simple and reliable syntax, modules and libraries, extensibility, scalability together make Python a suitable coding language for customizing larger applications. For example, Reddit which was originally written in Common Lips, was rewritten in Python in 2005. A large part of Youtube code is also written in Python.
9. **Operating Systems:**
   Python forms an integral part of Linux distributions.

Limitations of Python

1. Parallel processing can be done in Python but not as elegantly as done in some other languages (like JavaScript and Go Lang).
2. Being an interpreted language, Python is slow as compared to C/C++. Python is not a very good choice for those developing a high-graphic 3d game that takes up a lot of CPU.
3. As compared to other languages, Python is evolving continuously and there is little important documentation available for the language.
4. As of now, there are few users of Python as compared to those using C, C++ or Java.
5. It lacks true multiprocessor support.
6. It has very limited commercial support point.
7. Python is slower than C or C++ when it comes to computation heavy tasks and desktop applications.
8. It is difficult to pack up a big Python application into a single executable file. This makes it difficult to distribute Python to non-technical.

Using Python interpreter:
There are two ways to use the Python interpreter:

1. shell mode

In shell mode, you type Python expressions into the Python shell, and the interpreter immediately shows the result.(Open terminal and type python then you will get this prompt(>>>).Now try this example)

Example:

```
>>>print("5+2", 5+2)                                            7
>>>print("5-2",5-2)                                             3
>>>print("5*2",5*2)                                            10
>>>print("5/2",5/2.5)                                                2.5
>>>print("5%2",5%2)                                             1
>>>print("5**2",5**2)                                      25
>>>print("5//2",5//2)                                                2.0
>>>print("1+2-3*2= ", 1+2-3*2)                                 -3
```

This >>> is called the Python prompt. The interpreter uses the prompt to indicate that it is ready for instructions

Program mode

Write a source code in an editor. Steps to execute the python source code

Step 1: Open an editor.

Step 2: Write the instructions

Step 3: Save it as a file with the filename having the extension .py.

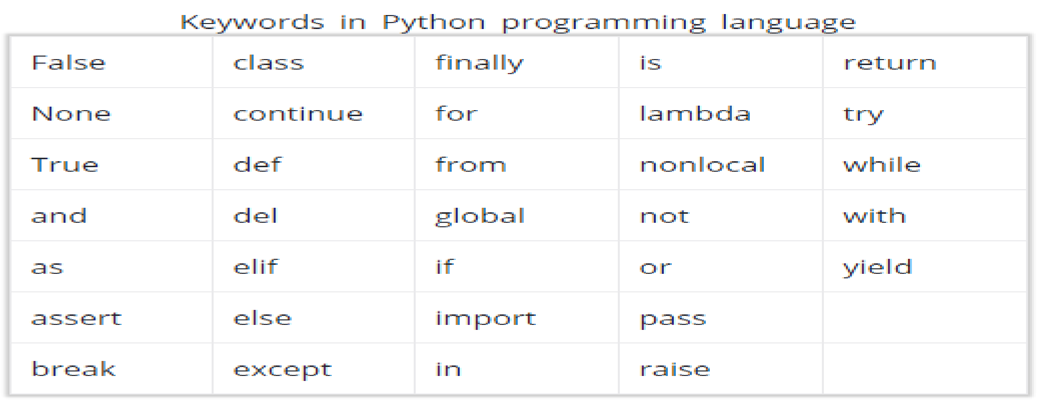
Step 4: Run the interpreter with the command python program_name.py or use IDLE to run the programs.

To execute the program at the command prompt, simply change your working directory to C:\Python34 (or move to the directory where you have saved Python) then type python program_name.py.

Keywords in Python

Keywords are the reserved words in Python. We cannot use a keyword as variable name, 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. 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 is given below.

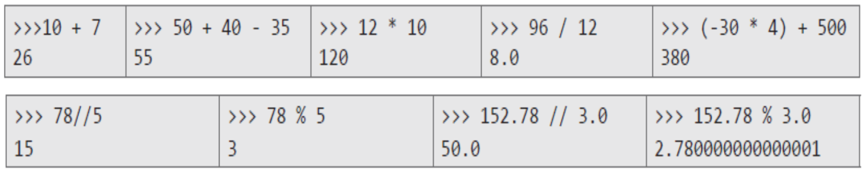


Keywords in Python programming language

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	

Literal Constants

A literal is a notation for representing a fixed value in source code. The value of a literal constant can be used directly in programs. For example, 7, 3.9, 'A', and "Hello" are literal constants.
Numbers refers to a numeric value. You can use four types of numbers in Python program- integers, long integers, floating point and complex numbers.
• Numbers like 5 or other whole numbers are referred to as integers. Bigger whole numbers are called long integers. For example, 535633629843L is a long integer.
• Numbers like are 3.23 and 91.5E-2 are termed as floating point numbers.
• Numbers of a + bi form (like -3 + 7i) are complex numbers.



```
>>>10 + 7
26
```

```
>>> 50 + 40 - 35
55
```

```
>>> 12 * 10
120
```

```
>>> 96 / 12
8.0
```

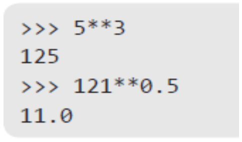
```
>>> (-30 * 4) + 500
380
```

```
>>> 78//5
15
```

```
>>> 78 % 5
3
```

```
>>> 152.78 // 3.0
50.0
```

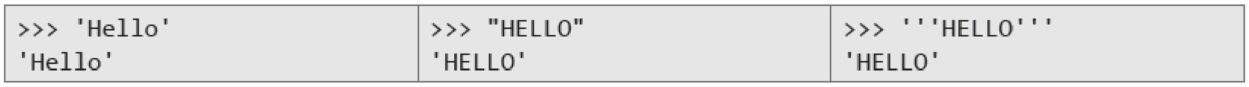
```
>>> 152.78 % 3.0
2.780000000000001
```



```
>>> 5**3
125
>>> 121**0.5
11.0
```

Strings
A string is a group of characters.
• Using Single Quotes ('): For example, a string can be written as 'HELLO'.
• Using Double Quotes ("): Strings in double quotes are exactly same as those in single quotes. Therefore, 'HELLO' is same as "HELLO".
• Using Triple Quotes (''' '''): You can specify multi-line strings using triple quotes. You can use as many single quotes and double quotes as you want in a string within triple quotes.

Examples



```
>>> 'Hello'
'Hello'
```

```
>>> "HELLO"
'HELLO'
```

```
>>> '''HELLO'''
'HELLO'
```

Python Statement

Instructions that a Python interpreter can execute are called statements. For example, a = 1 is an assignment statement.

Multi-line statement
In Python, end of a statement is marked by a newline character. But we can make a statement extend over multiple lines with the line continuation character (\).
Example:

```
a = 1 + 2 + 3 + \
4 + 5 + 6 + \
7 + 8 + 9
```

Escape Sequences

Some characters (like ", \) cannot be directly included in a string. Such characters must be escaped by placing a backslash before them.

Example:

```
>>> print("The boy replies, \"My name is Aaditya.\"")
The boy replies, "My name is Aaditya."
```

Escape Sequence	Purpose	Example	Output
`\\`	Prints Backslash	`print("\\")`	`\`
`\'`	Prints single-quote	`print("\'")`	`'`
`\"`	Prints double-quote	`print("\"")`	`"`
`\a`	Rings bell	`print("\a")`	Bell rings
`\f`	Prints form feed character	`print("Hello\fWorld")`	Hello World
`\n`	Prints newline character	`print("Hello\nWorld")`	Hello World
`\t`	Prints a tab	`print( "Hello\tWorld")`	Hello World
`\o`	Prints octal value	`print("\o56")`	.
`\x`	Prints hex value	`print("\x87")`	+

Raw Strings

If you want to specify a string that should not handle any escape sequences and want to display exactly as specified then you need to specify that string as a raw string. A raw string is specified by prefixing r or R to the string.

Example

```
>>> print(R "What\'s your name?")
What\'s your name?
```

Variables and Identifiers

Variable means its value can vary. You can store any piece of information in a variable. Variables are nothing but just parts of your computer's memory where information is stored. To be identified easily, each variable is given an appropriate name.

Identifiers are names given to identify something. This something can be a variable, function, class, module or other object. For naming any identifier, there are some basic rules like:

- The first character of an identifier must be an underscore ('_') or a letter (upper or lowercase).
- The rest of the identifier name can be underscores ('_'), letters (upper or lowercase), or digits (0-9).
- Identifier names are case-sensitive. For example, myvar and myVar are not the same.
- Punctuation characters such as @, $, and % are not allowed within identifiers.

Examples of valid identifier names are sum, __my_var, num1, r, var_20, First, etc.

Examples of invalid identifier names are 1num, my-var, %check, Basic Sal, H#R&A, etc.

Assigning or Initializing Values to Variables

In Python, programmers need not explicitly declare variables to reserve memory space. The declaration is done automatically when a value is assigned to the variable using the equal sign (=). The operand on the left side of equal sign is the name of the variable and the operand on its right side is the value to be stored in that variable.

Example

num=7

```
amt=123.45
code='A'
pi=3.1415926536
population=100000000000000
msg="hi"
print "num",num
print "amt",amt
print "code",code
print "pi",pi
print "population",population
print "msg",msg
```

Output

```
num 7
amt 123.45
code A
pi 3.1415926536
population 100000000000000
msg hi
```

Multiple assignments

In Python, multiple assignments can be made in a single statement as follows:

Ex: a, b, c = 5, 3.2, "Hello"

If we want to assign the same value to multiple variables at once, we can do this as:

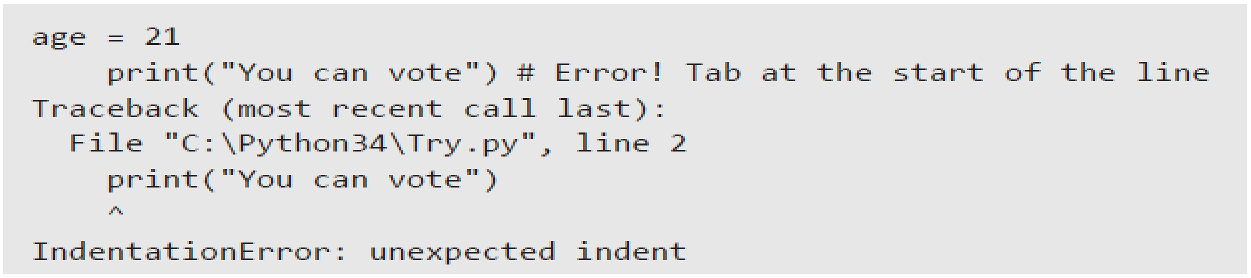
Ex: x = y = z = "same"

This assigns the "same" string to all the three variables.

Python Indentation

Whitespace at the beginning of the line is called indentation. These whitespaces or the indentation are very important in Python. In a Python program, the leading whitespace including spaces and tabs at the beginning of the logical line determines the indentation level of that logical line.

Example



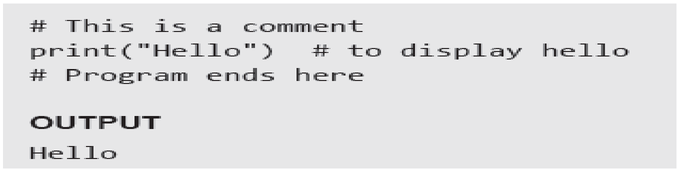
```
age = 21
    print("You can vote") # Error! Tab at the start of the line
Traceback (most recent call last):
  File "C:\Python34\Try.py", line 2
    print("You can vote")
    ^
IndentationError: unexpected indent
```

Python Comments

Comments are the non-executable statements in a program. They are just added to describe the statements in the program code. Comments make the program easily readable and understandable by the programmer as well as other users who are seeing the code. The interpreter simply ignores the comments.

In Python, a hash sign (#) that is not inside a string literal begins a comment. All characters following the # and up to the end of the line are part of the comment

Example



```
# This is a comment
print("Hello")   # to display hello
# Program ends here

OUTPUT
Hello
```

Multi-line comments
If we have comments that extend multiple lines, We can use triple quotes, either ''' or """.
These triple quotes are generally used for multi-line strings. But they can be used as multi-line comment as well.
Example

```
"""This is also a
perfect example of
multi-line comments"""
```

Python Output Using print() function

We use the print() function to output data to the standard output device (screen).

Example

```
print('This sentence is output to the screen')
# Output: This sentence is output to the screen
a = 5
print('The value of a is', a)
# Output: The value of a is 5
```

Output formatting

Sometimes we would like to format our output to make it look attractive. This can be done by using the str.format() method. This method is visible to any string object.
Example 1:

```
x = 5; y = 10
print('The value of x is {} and y is {}'.format(x,y))
```

Output: The value of x is 5 and y is 10

Here the curly braces {} are used as placeholders. We can specify the order in which it is printed by using numbers (tuple index).
Example 2:

```
print('I love {0} and {1}'.format('bread','butter'))
# Output: I love bread and butter
print('I love {1} and {0}'.format('bread','butter'))
```

# Output: I love butter and bread

We can even use keyword arguments to format the string.
Example 3:

```
print('Hello {name}, {greeting}'.format(greeting = 'Goodmorning', name = 'John'))
```

#Output: Hello John, Goodmorning

Example 4:

```
x = 5.3333333333333
print "x value is {}".format(round(x,2))
```

#Output: x value is 5.33

Python Input

To take input from the users, Python makes use of the input() function. The input() function prompts the user to provide some information on which the program can work and give the result.

- input( ) – input function is used to take input from the user
- raw_input( ) – it takes the input in the form of a string and reads the data line by line or line at a time

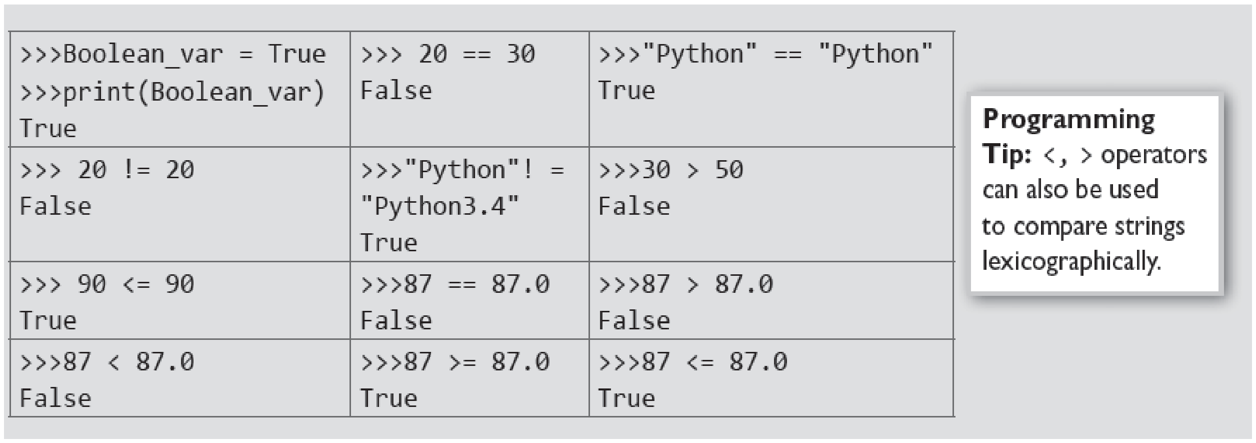
Example:

```
name=raw_input("What is your name")
age=input("how old are you")
print "hi {} your age is {}".format(name,age)
```

Data Type Boolean

Boolean is another data type in Python. A variable of Boolean type can have one of the two values- True or False. Similar to other variables, the Boolean variables are also created while we assign a value to them or when we use a relational operator on them.

Example



`>>>Boolean_var = True`
`>>>print(Boolean_var)`
`True`
	`>>> 20 == 30`
	`False`
		`>>>"Python" == "Python"`
		`True`

`>>> 20 != 20`
`False`
	`>>>"Python"! =`
	`"Python3.4"`
	`True`
		`>>>30 > 50`
		`False`

`>>> 90 <= 90`
`True`
	`>>>87 == 87.0`
	`False`
		`>>>87 > 87.0`
		`False`

`>>>87 < 87.0`
`False`
	`>>>87 >= 87.0`
	`True`
		`>>>87 <= 87.0`
		`True`

Programming Tip: <, > operators can also be used to compare strings lexicographically.

Python Operators

Operators are special symbols in Python that carry out arithmetic or logical computation. The value that the operator operates on is called the operand.

Arithmetic operators

Arithmetic operators are used to perform mathematical operations like addition, subtraction, multiplication etc.

Operator	Description	Example	Output
+	Addition: Adds the operands	`>>> print(a + b)`	300
-	Subtraction: Subtracts operand on the right from the operand on the left of the operator	`>>> print(a – b)`	-100
*	Multiplication: Multiplies the operands	`>>> print(a * b)`	20000
/	Division: Divides operand on the left side of the operator with the operand on its right. The division operator returns the quotient.	`>>> print(b / a)`	2.0
%	Modulus: Divides operand on the left side of the operator with the operand on its right. The modulus operator returns the remainder.	`>>> print(b % a)`	0

//	Floor Division: Divides the operands and returns the quotient. It also removes the digits after the decimal point. If one of the operands is negative, the result is floored (i.e.,rounded away from zero towards negative infinity).	`>>> print(12//5)`
		`>>> print( 12.0//5.0)`
		`>>> print(-19//5)`
		`>>> print(-20.0//3)`
			2
			2.0
			-4
			-7.0

**	Exponent: Performs exponential calculation, that is, raises operand on the right side to the operand on the left of the operator.	`>>> print(a**b)`	100^{200}

Example

```
x = 15
y = 4
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)
```

Output

```
('x + y =', 19)
('x - y =', 11)
('x * y =', 60)
('x // y =', 3)
('x ** y =', 50625)
```

Comparison operators

Comparison operators are used to compare values. It either returns True or False according to the condition.

Operator	Description	Example	Output
==	Returns True if the two values are exactly equal.	`>>> print(a == b)`	False
!=	Returns True if the two values are not equal.	`>>> print(a != b)`	True
>	Returns True if the value at the operand on the left side of the operator is greater than the value on its right side.	`>>> print(a > b)`	False
<	Returns True if the value at the operand on the right side of the operator is greater than the value on its left side.	`>>> print(a < b)`	True
>=	Returns True if the value at the operand on the left side of the operator is either greater than or equal to the value on its right side.	`>>> print(a >= b)`	False
<=	Returns True if the value at the operand on the right side of the operator is either greater than or equal to the value on its left side.	`>>> print(a <= b)`	True

Example

```
x = 10
y = 12
print('x > y  is',x>y)
print('x < y  is',x<y)
```

Output

```
('x > y  is', False)
('x < y  is', True)
```

Logical operators

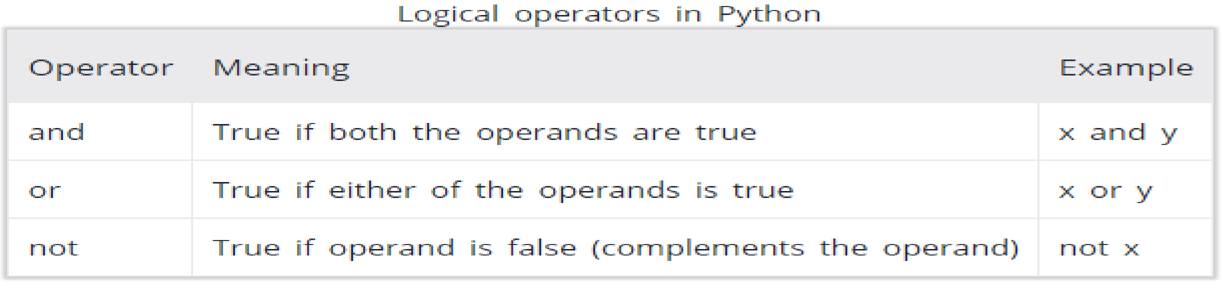
Logical operators are the and, or, not operators. Based on Boolean Algebra Returns results as either True or False

Logical AND (&&) operator is used to simultaneously evaluate two conditions or expressions with relational operators. If expressions on both the sides (left and right side) of the logical operator are true, then the whole expression is true. For example, If we have an expression (a>b) && (b>c), then the whole expression is true only if both expressions are true. That is, if b is greater than a and c.

Logical OR (| |) operator is used to simultaneously evaluate two conditions or expressions with relational operators. If one or both the expressions of the logical operator is true, then the whole expression is true. For example, If we have an expression (a>b) | | (b>c), then the whole expression is true if either b is greater than a or b is greater than c.

Logical not (!) operator takes a single expression and negates the value of the expression. Logical NOT produces a zero if the expression evaluates to a non-zero value and produces a 1 if the expression produces a zero. In other words, it just reverses the value of the expression. For example, a = 10, b b = !a; Now, the value of b = 0. The value of a is not zero, therefore, !a = 0. The value of !a is assigned to b, hence, the result.

Logical operators in Python



Operator	Meaning	Example
and	True if both the operands are true	x and y
or	True if either of the operands is true	x or y
not	True if operand is false (complements the operand)	not x

Example

```
x = True
y = False
# Output: x and y is False
print('x and y is',x and y)
# Output: x or y is True
print('x or y is',x or y)
# Output: not x is False
print('not x is',not x)
```

Assignment operators

Assignment operators are used in Python to assign values to variables. a = 5 is a simple assignment operator that assigns the value 5 on the right to the variable a on the left.

Operator	Example	Equivatent to
=	x = 5	x = 5
+=	x += 5	x = x + 5
-=	x -= 5	x = x - 5
*=	x *= 5	x = x * 5
/=	x /= 5	x = x / 5
%=	x %= 5	x = x % 5
//=	x //= 5	x = x // 5
**=	x **= 5	x = x ** 5
&=	x &= 5	x = x & 5
|=	x |= 5	x = x | 5
^=	x ^= 5	x = x ^ 5
>>=	x >>= 5	x = x >> 5
<<=	x <<= 5	x = x << 5

Bitwise Operator

As the name suggests, bitwise operators perform operations at the bit level. These operators include bitwise AND, bitwise OR, bitwise XOR, and shift operators. Bitwise operators expect their operands to be of integers and treat them as a sequence of bits.

The truth tables of these bitwise operators are given below.

A	B	A&B	A	B	A|B	A	B	A^B	A	!A
0	0	0	0	0	0	0	0	0	0	1
0	1	0	0	1	1	0	1	1	1	0
1	0	0	1	0	1	1	0	1		
1	1	1	1	1	1	1	1	0		

Shift Operators

Python supports two bitwise shift operators. They are shift left (<<) and shift right (>>). These operations are used to shift bits to the left or to the right. The syntax for a shift operation can be given as follows:

```
if we have x = 0001 1101, then
x << 1 gives result = 0011 1010
if we have x = 0001 1101, then
x << 4 gives result = 1010 0000
if we have x = 0001 1101, then
x >> 1 gives result = 0000 1110.
Similarly, if we have x = 0001 1101 then
x << 4 gives result = 0000 0001
```

Unary Operators

Unary operators act on single operands. Python supports unary minus operator. Unary minus operator is strikingly different from the arithmetic operator that operates on two operands and subtracts the second operand from the first operand. When an operand is preceded by a minus sign, the unary operator negates its value.

For example, if a number is positive, it becomes negative when preceded with a unary minus operator. Similarly, if the number is negative, it becomes positive after applying the unary minus operator. Consider the given example.
b = 10 a = -(b)
The result of this expression, is a = -10, because variable b has a positive value. After applying unary minus operator (-) on the operand b, the value becomes -10, which indicates it as a negative value.

Identity operators:
is Operator: Returns true if operands or values on both sides of the operator point to the same object and false otherwise.
is not Operator: Returns true if operands or values on both sides of the operator does not point to the same object and false otherwise.

Identity operators in Python

Operator	Meaning	Example
Is	True if the operands are identical (refer to the same object)	x is True
is not	True if the operands are not identical (do not refer to the same object)	x is not True

Example

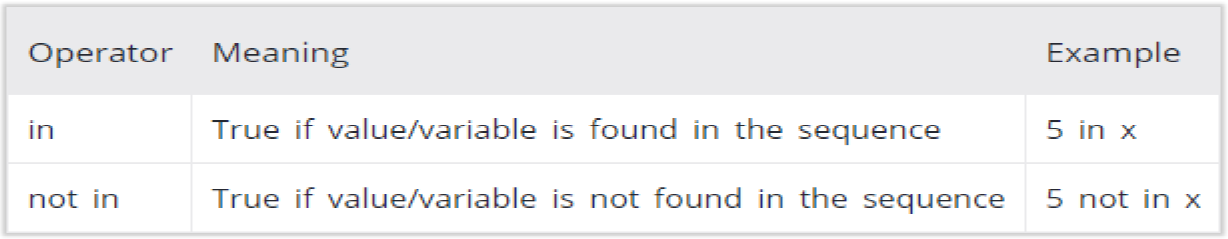
```
x1 = 5
y1 = 5
x2 = 'Hello'
y2 = 'Hello'
x3 = [1,2,3]
y3 = [1,2,3]
# Output: False
print(x1 is not y1)
# Output: True
print(x2 is y2)
# Output: False
print(x3 is y3)
```

Here, we see that x1 and y1 are integers of same values, so they are equal as well as identical. Same is the case with x2 and y2 (strings).

But x3 and y3 are list. They are equal but not identical. Since list are mutable (can be changed), interpreter locates them separately in memory although they are equal.

Membership operators
Python supports two types of membership operators–in and not in. These operators, test for membership in a sequence such as strings, lists, or tuples.

in Operator: The operator returns true if a variable is found in the specified sequence and false otherwise.
not in Operator: The operator returns true if a variable is not found in the specified sequence and false otherwise.



Operator	Meaning	Example
in	True if value/variable is found in the sequence	5 in x
not in	True if value/variable is not found in the sequence	5 not in x

Example

```
x = 'Hello world'
y = {1:'a',2:'b'}
# Output: True
print('H' in x)
# Output: True
print('hello' not in x)
# Output: True
print(1 in y)
# Output: False
print('a' in y)
```

Here, 'H' is in x but 'hello' is not present in x (remember, Python is case sensitive). Similary, 1 is key and 'a' is the value in dictionary y. Hence, 'a' in y returns False.

Expressions
An expression is any legal combination of symbols (like variables, constants and operators) that represents a value. In Python, an expression must have at least one operand (variable or constant) and can have one or more operators. On evaluating an expression, we get a value. Operand is the value on which operator is applied.
Constant Expressions: One that involves only constants. Example: 8 + 9 – 2
Integral Expressions: One that produces an integer result after evaluating the expression. Example:
a = 10
• **Floating Point Expressions:** One that produces floating point results. Example: a * b / 2
• **Relational Expressions:** One that returns either true or false value. Example: c = a>b
• **Logical Expressions**: One that combines two or more relational expressions and returns a value as True or False. Example: a>b && y! = 0
• **Bitwise Expressions:** One that manipulates data at bit level. Example: x = y&z
• **Assignment Expressions:** One that assigns a value to a variable. Example: c = a + b or c = 10

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
* / % //	Multiply, divide, modulo and floor division

+ -	Addition and subtraction
>> <<	Right and left bitwise shift
&	Bitwise 'AND'
^ |	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

NOTE: There is an exception to the left-to-right rule. When two ** operators share an operand, the operators execute right-to-left. For example, the expression 2**3**4 is evaluated as 2**(3**4).

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 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
```

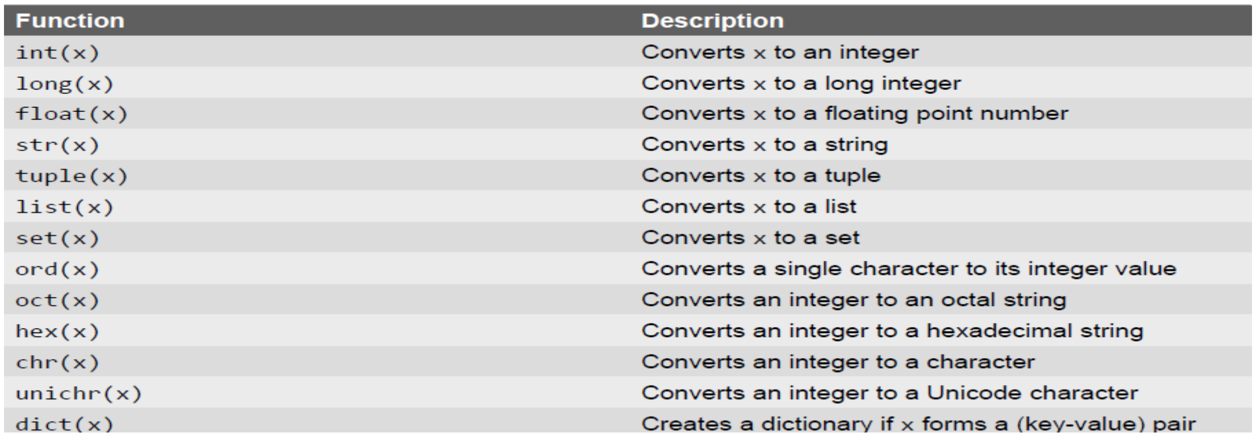
Type Conversion

In Python, it is just not possible to complete certain operations that involves different types of data. For example, it is not possible to perform "2" + 4 since one operand is an integer and the other is of string type.

Example

```
>>>"20"+"30"
'2030'
```

```
>>> int("2") + int("3")
5
```



Function	Description
int(x)	Converts x to an integer
long(x)	Converts x to a long integer
float(x)	Converts x to a floating point number
str(x)	Converts x to a string
tuple(x)	Converts x to a tuple
list(x)	Converts x to a list
set(x)	Converts x to a set
ord(x)	Converts a single character to its integer value
oct(x)	Converts an integer to an octal string
hex(x)	Converts an integer to a hexadecimal string
chr(x)	Converts an integer to a character
unichr(x)	Converts an integer to a Unicode character
dict(x)	Creates a dictionary if x forms a (key-value) pair

Built-in function : type( )

It is used to identify the type of the object

Example:

```
int_a =10
print("Type of int_a:",type(int_a))
str_b="Hello"
print("Type of str_b:", type(str_b))
list_c=[ ]
print("Type of list_c:",type(list_c))
```

Output:

```
('Type of int_a:', <type 'int'>)
('Type of str_b:', <type 'str'>)
('Type of list_c:', <type 'list'>)>
```