TATA ELXSI

Python Scripting

Learning & Development Team

Monday, November 2, 2020

TATA ELXSI

Module - 1

Disclaimer

- This material is developed for in house training at **TATA ELXSI**.
- *TATA ELXSI is not responsible for any errors or omissions, or for the results obtained from the use of this information. All information in this material is provided "as is", with no guarantee of completeness, timeliness or of the results obtained from the use of this information...*
- *This is a training material and the information contained herein is not intended to be a source of advice or research outcome with respect to the material presented, and the information and/or documents/examples contained in this material do not constitute authors or TATA ELXSI's advice.*
- The training material therefore was solely developed for educational purposes. Currently, commercialization of the prototype does not exist, nor is the prototype available for general usage.
- **TATA ELXSI**, its staff, its students or any other participants **only** can use it as a part of training. This material should not be found on any website.
- For the preparation of this material we have referred from the below mentioned links or books.
- Excerpts of these material has been taken where there is no copy right infringements.

Agenda

- Introduction to Python
- Working with Python
- Salient features of Python
- interactive "shell"
- basic types: numbers, strings
- Operators
- Built-in functions
- Program Flow Constructs

History

- Python implementation began in December 1989 by Guido van Rossum at Centrum Wiskunde & Informatica (CWI) in the Netherlands.
- December 1989, I was looking for a "hobby" programming project that would keep me occupied during the week around Christmas.
- I chose Python as a working title for the project, being in a slightly irreverent mood (and a big fan of Monty Python's Flying Circus).

Introduction

- Python is a programming language that lets you work more quickly and integrate your systems more effectively.
  - ➢ Python is an easy to learn,
  - ➢ very clear, readable syntax
  - ➢ exception-based error handling
  - ➢ very high level dynamic data types

- It has efficient high-level data structures and a simple but effective approach to object-oriented programming.
  - ➢ OOP is optional

Main use cases

DATA SCIENCE

DATA ANALYTICS

ARTIFICIAL INTELLIGENCE, DEEP LEARNING

ENTERPRISE APPLICATION

WEB DEVELOPMENT(WEB APPLICATIONS. YOUTUBE, INSTAGRAM, PINTEREST, SURVEYMONKEY ARE ALL BUILT-IN PYTHON.)

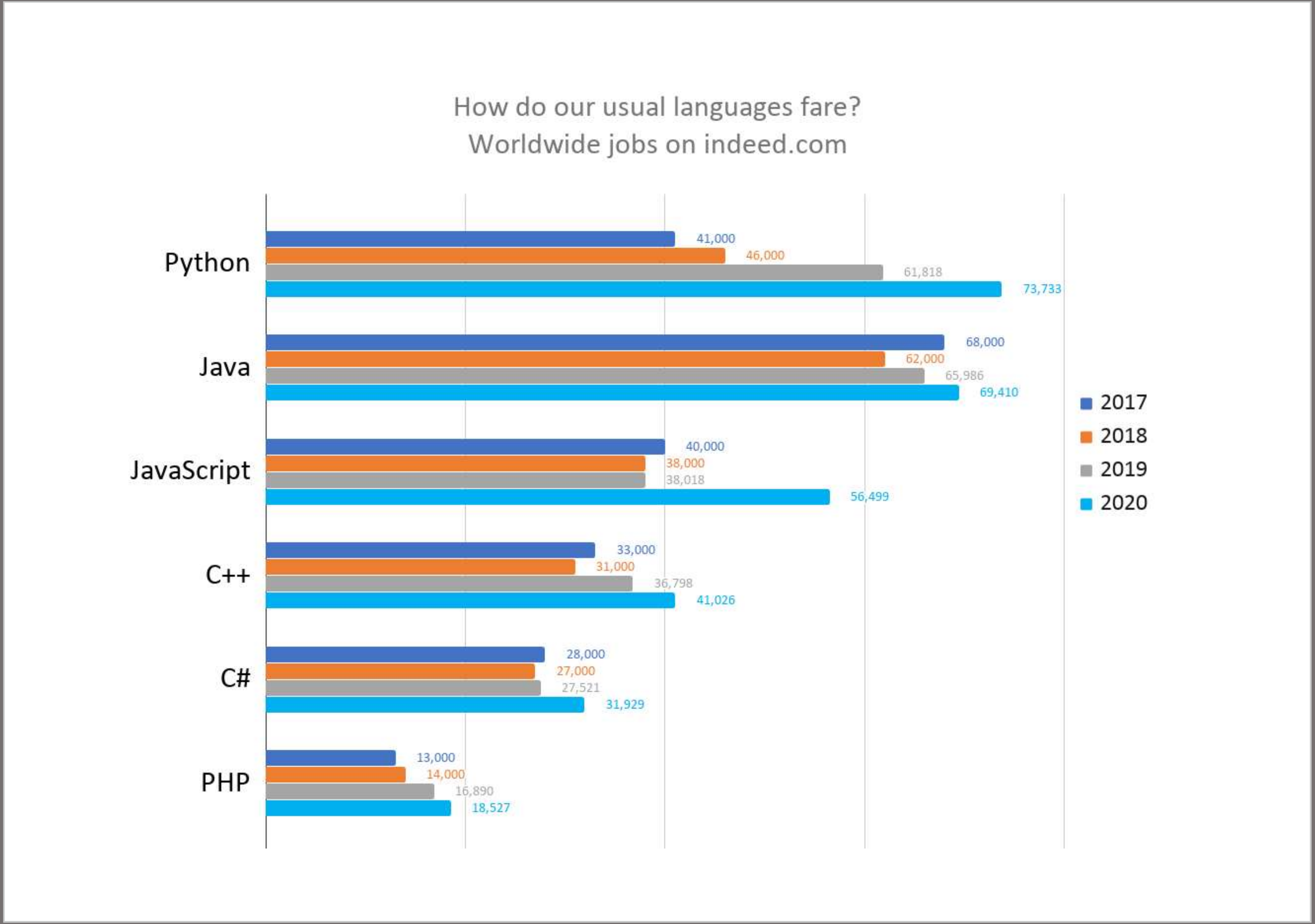
Who use python today?

- Google
- Facebook
- Instagram
- Spotify
- Quora
- Netflix
- Dropbox
- Reddit

Python’s core philosophy

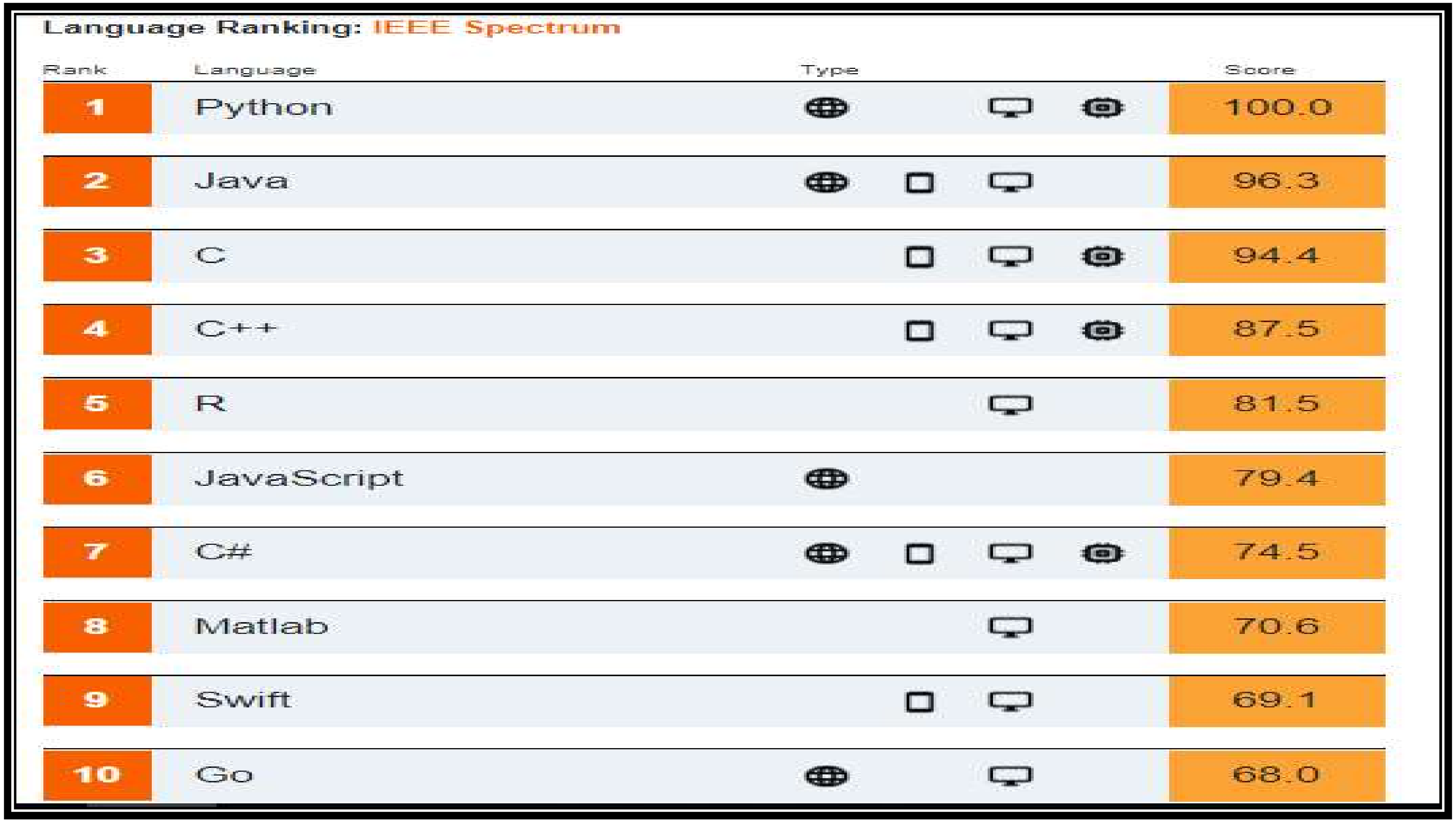
- **The language's core philosophy is summarized in the document The Zen of Python (PEP 20)**

  - Beautiful is better than ugly.
  - Explicit is better than implicit.
  - Simple is better than complex.
  - Complex is better than complicated.
  - Readability counts.



Most popular in 2020

Top 10 Languages by IEEE Spectrum



Language Ranking: IEEE Spectrum

Rank	Language	Type				Score
1	Python	🌐		🖥	⚙	100.0
2	Java	🌐	▢	🖥		96.3
3	C		▢	🖥	⚙	94.4
4	C++		▢	🖥	⚙	87.5
5	R			🖥		81.5
6	JavaScript	🌐				79.4
7	C#	🌐	▢	🖥	⚙	74.5
8	Matlab			🖥		70.6
9	Swift		▢	🖥		69.1
10	Go	🌐		🖥		68.0

Introduction

- Python's elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for
  - ➢ Scripting and
  - ➢ Rapid application development in many areas on most platforms.

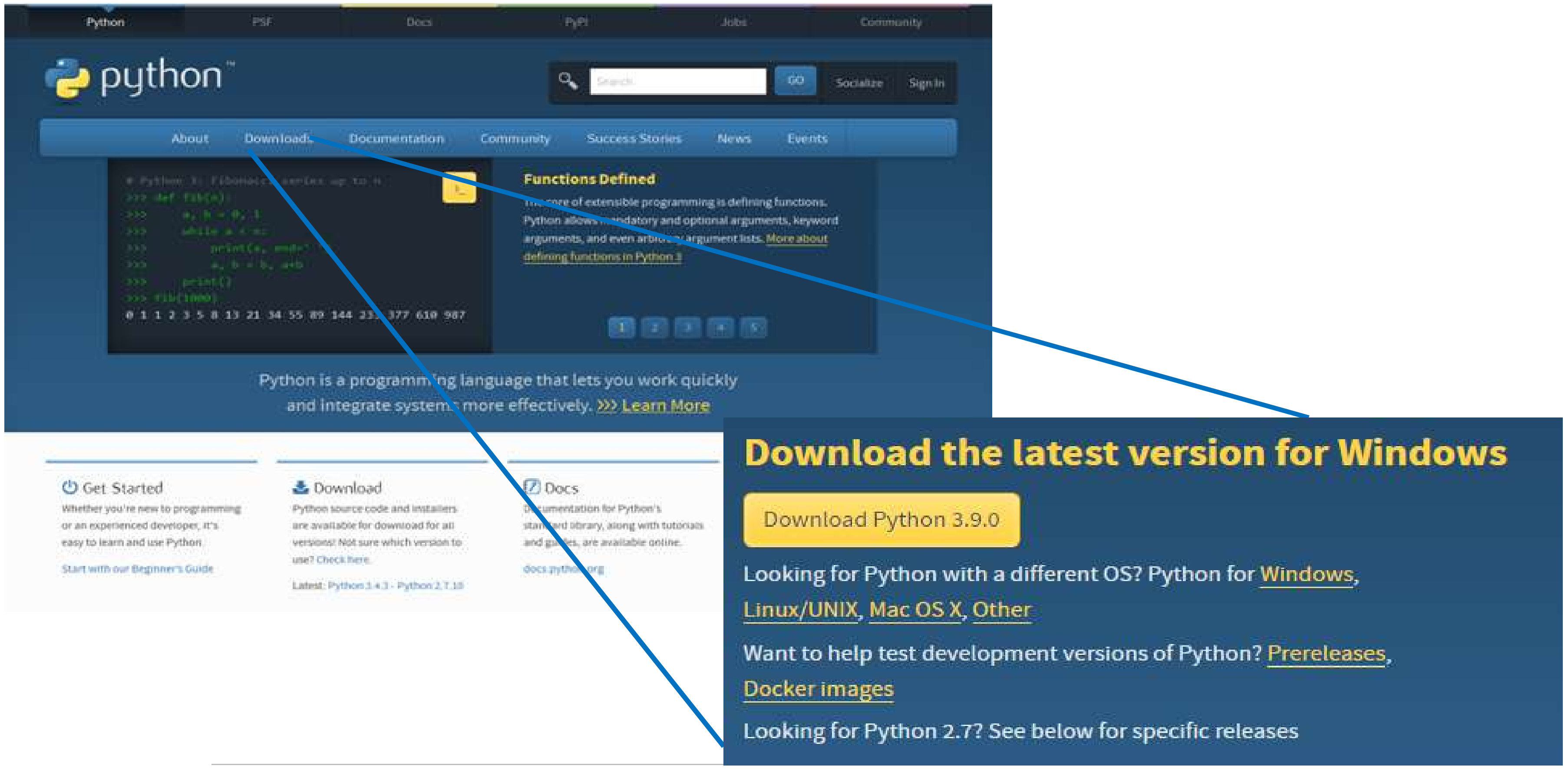
- Python runs everywhere.

- Python can be integrated with another language.

Where to get Python

- To download the appropriate file for your computer from

http://www.python.org/download



```
# Python 3: Fibonacci series up to n
>>> def fib(n):
>>>     a, b = 0, 1
>>>     while a < n:
>>>         print(a, end=' ')
>>>         a, b = b, a+b
>>>     print()
>>> fib(1000)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
```

Functions Defined

The core of extensible programming is defining functions. Python allows mandatory and optional arguments, keyword arguments, and even arbitrary argument lists. More about defining functions in Python 3

Python is a programming language that lets you work quickly and integrate systems more effectively. >>> Learn More

Get Started
Whether you're new to programming or an experienced developer, it's easy to learn and use Python.
Start with our Beginner's Guide

Download
Python source code and installers are available for download for all versions! Not sure which version to use? Check here.
Latest: Python 3.4.3 - Python 2.7.10

Docs
Documentation for Python's standard library, along with tutorials and guides, are available online.
docs.python.org

Download the latest version for Windows

Download Python 3.9.0

Looking for Python with a different OS? Python for Windows, Linux/UNIX, Mac OS X, Other

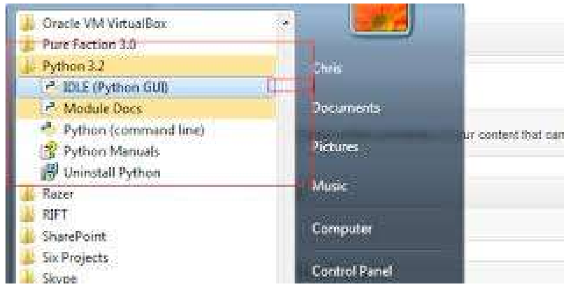
Want to help test development versions of Python? Prereleases, Docker images

Looking for Python 2.7? See below for specific releases

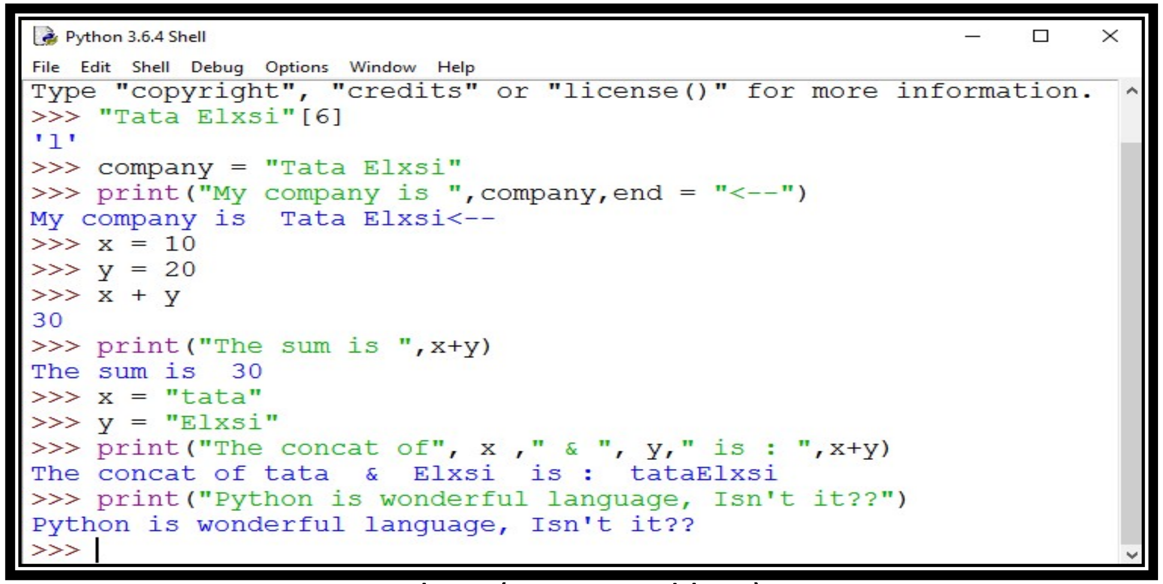
Interactive “Shell”

- Great for learning the language
- Great for experimenting with the library
- Great for testing your own modules
- Type statements or expressions at prompt:

```
>>> print "Hello, world"

Hello, world

>>> x = 12**2

>>> x/2

72

>>> # this is a comment
```



Two variations: IDLE (GUI),



```
Type "copyright", "credits" or "license()" for more information.
>>> "Tata Elxsi"[6]
'l'
>>> company = "Tata Elxsi"
>>> print("My company is ",company,end = "<--")
My company is  Tata Elxsi<--
>>> x = 10
>>> y = 20
>>> x + y
30
>>> print("The sum is ",x+y)
The sum is  30
>>> x = "tata"
>>> y = "Elxsi"
>>> print("The concat of", x ," & ", y," is : ",x+y)
The concat of tata  &  Elxsi  is :  tataElxsi
>>> print("Python is wonderful language, Isn't it??")
Python is wonderful language, Isn't it??
>>> 
```

python (command line)

Creating and Running Programs

- Go into IDLE if you are not already.
- Go to File then New Window.

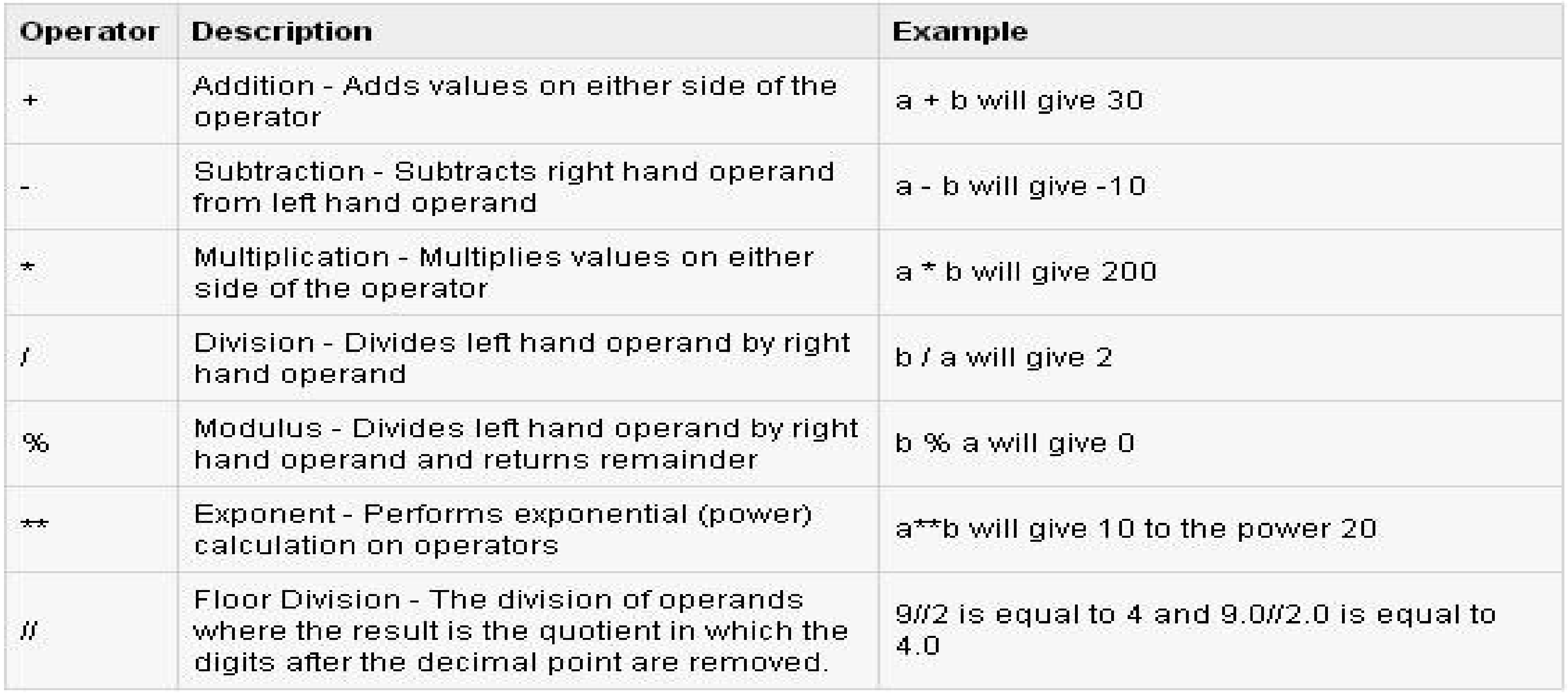
```
#!/usr/bin/python
print("Jack and Jill went up a hill“)

print("to fetch a pail of water;“)

print("Jack fell down, and broke his
    crown,“)

print("and Jill came tumbling after.“)
```

OutPut:

```
Jack and Jill went up a hill
to fetch a pail of water;
Jack fell down, and broke his
crown,
and Jill came tumbling after.
```

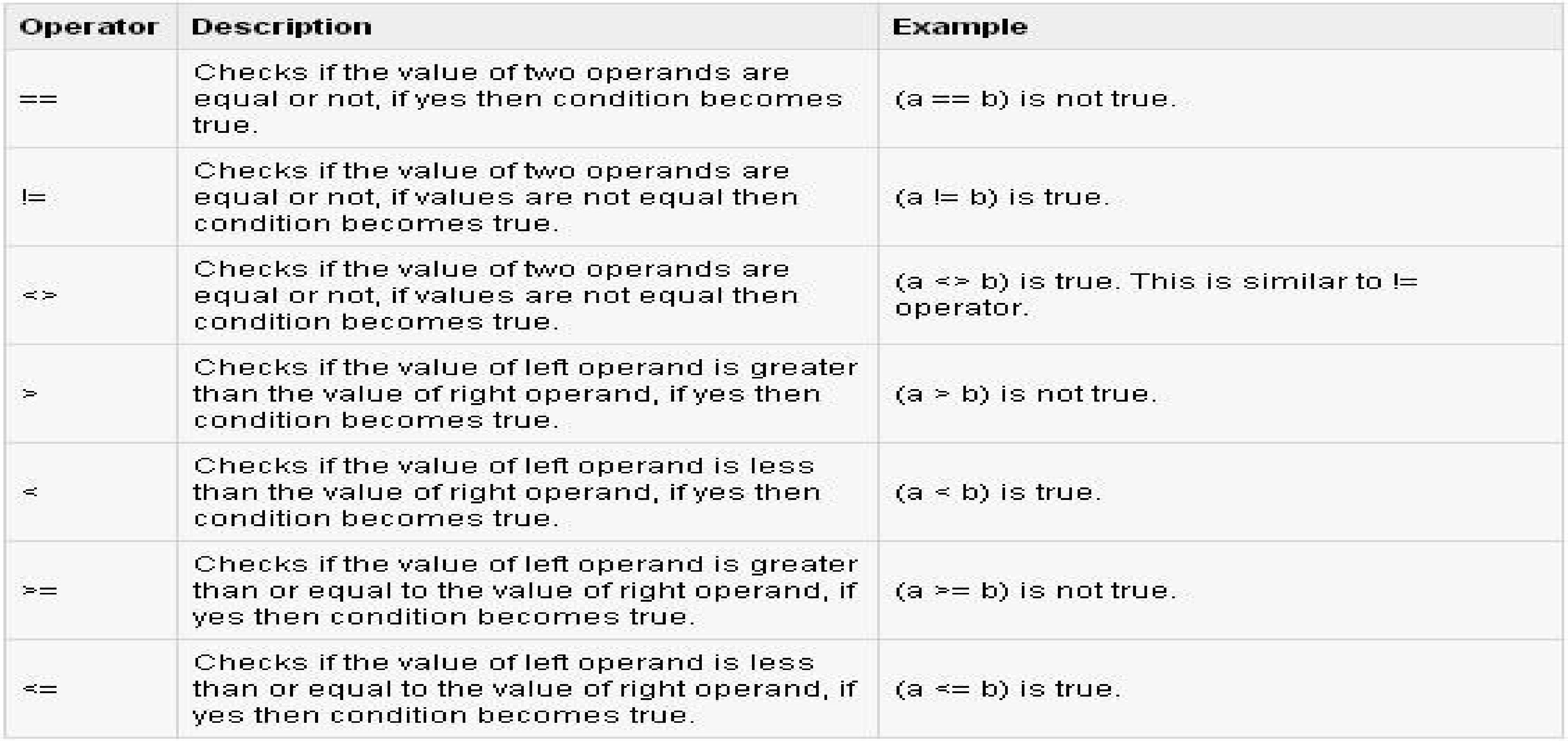
Python Arithmetic Operators

- Assume variable ‘a’ holds 10 and variable ‘b’ holds 20



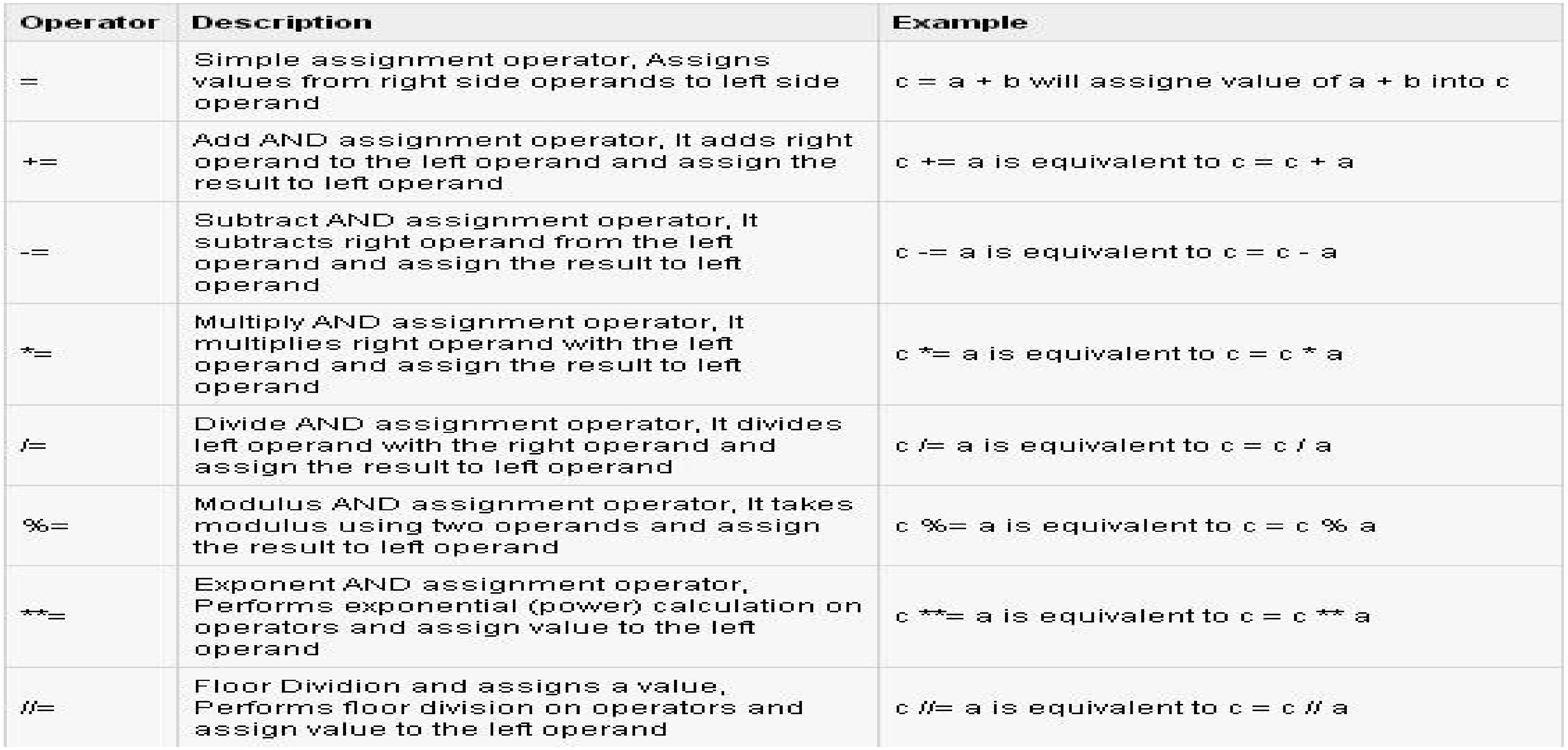
Operator	Description	Example
+	Addition - Adds values on either side of the operator	a + b will give 30
-	Subtraction - Subtracts right hand operand from left hand operand	a - b will give -10
*	Multiplication - Multiplies values on either side of the operator	a * b will give 200
/	Division - Divides left hand operand by right hand operand	b / a will give 2
%	Modulus - Divides left hand operand by right hand operand and returns remainder	b % a will give 0
**	Exponent - Performs exponential (power) calculation on operators	a**b will give 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.	9//2 is equal to 4 and 9.0//2.0 is equal to 4.0

Python Comparison Operators



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

Python Assignment Operators

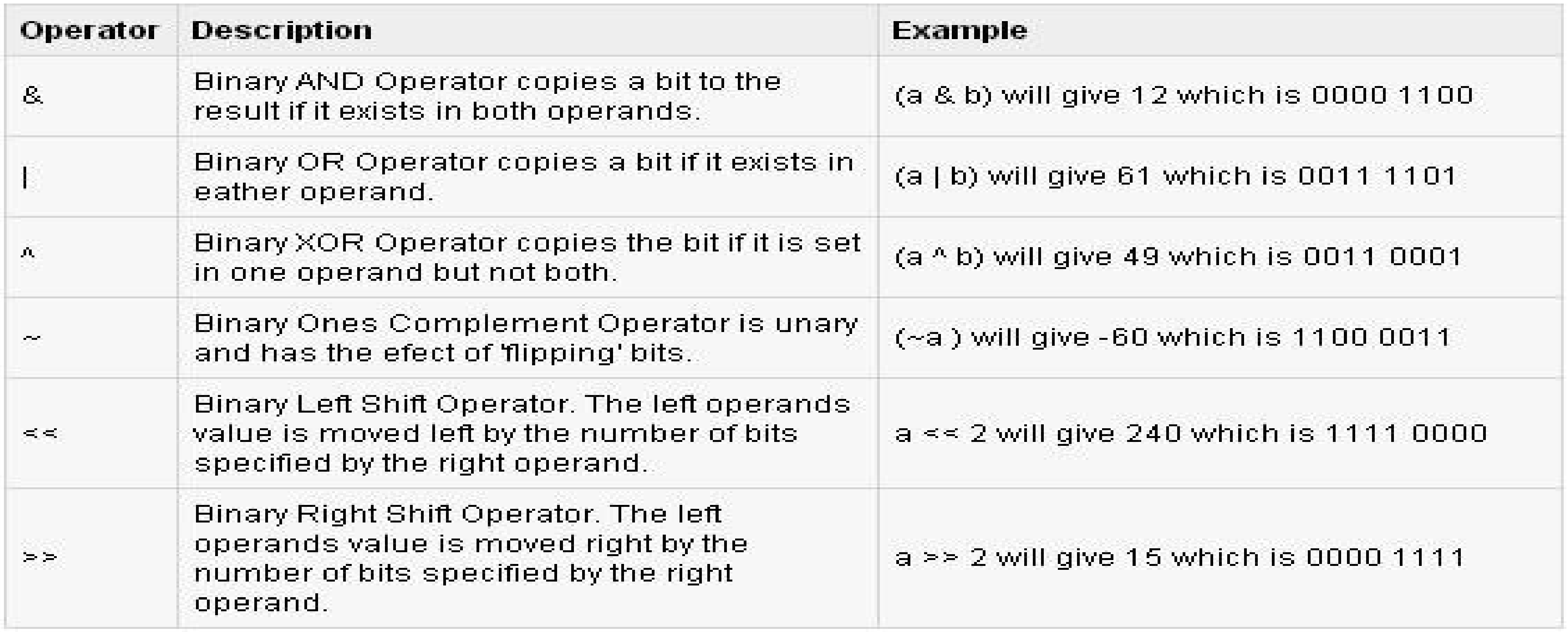


Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	c = a + b will assigne value of a + b into c
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	c += a is equivalent to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

TATA ELXSI

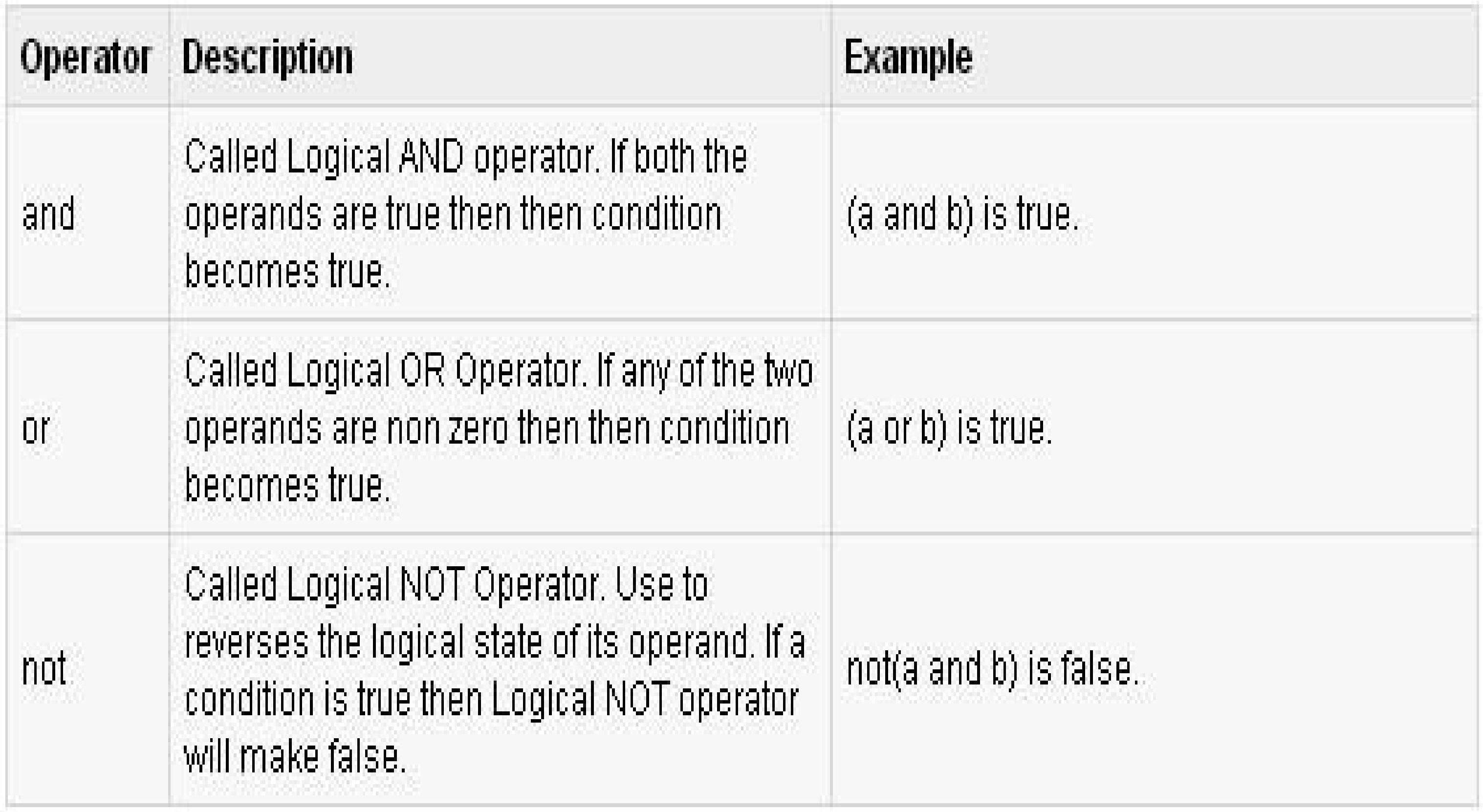
Python Bitwise Operators

- Assume if a = 60; and b = 13; Now in binary format they will be as follows:
- a = 0011 1100
- b = 0000 1101



Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(a & b) will give 12 which is 0000 1100
|	Binary OR Operator copies a bit if it exists in eather operand.	(a | b) will give 61 which is 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(a ^ b) will give 49 which is 0011 0001
~	Binary Ones Complement Operator is unary and has the efect of 'flipping' bits.	(~a) will give -60 which is 1100 0011
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	a << 2 will give 240 which is 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	a >> 2 will give 15 which is 0000 1111

Python Logical Operators

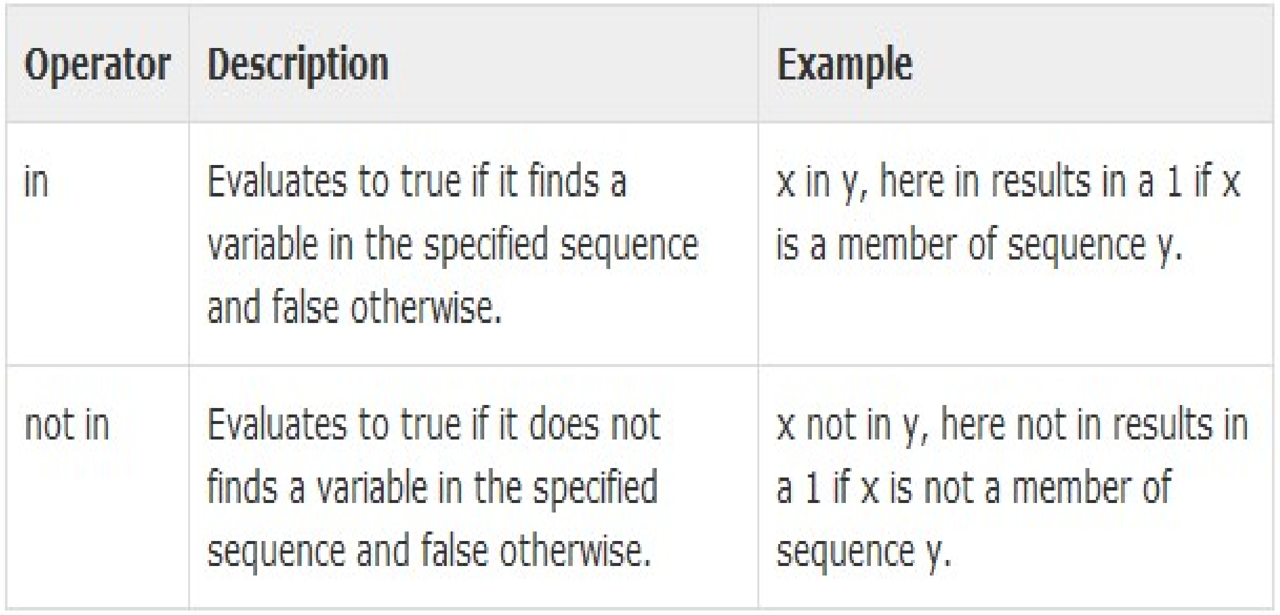


Operator	Description	Example
and	Called Logical AND operator. If both the operands are true then then condition becomes true.	(a and b) is true.
or	Called Logical OR Operator. If any of the two operands are non zero then then condition becomes true.	(a or b) is true.
not	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	not(a and b) is false.

TATA ELXSI

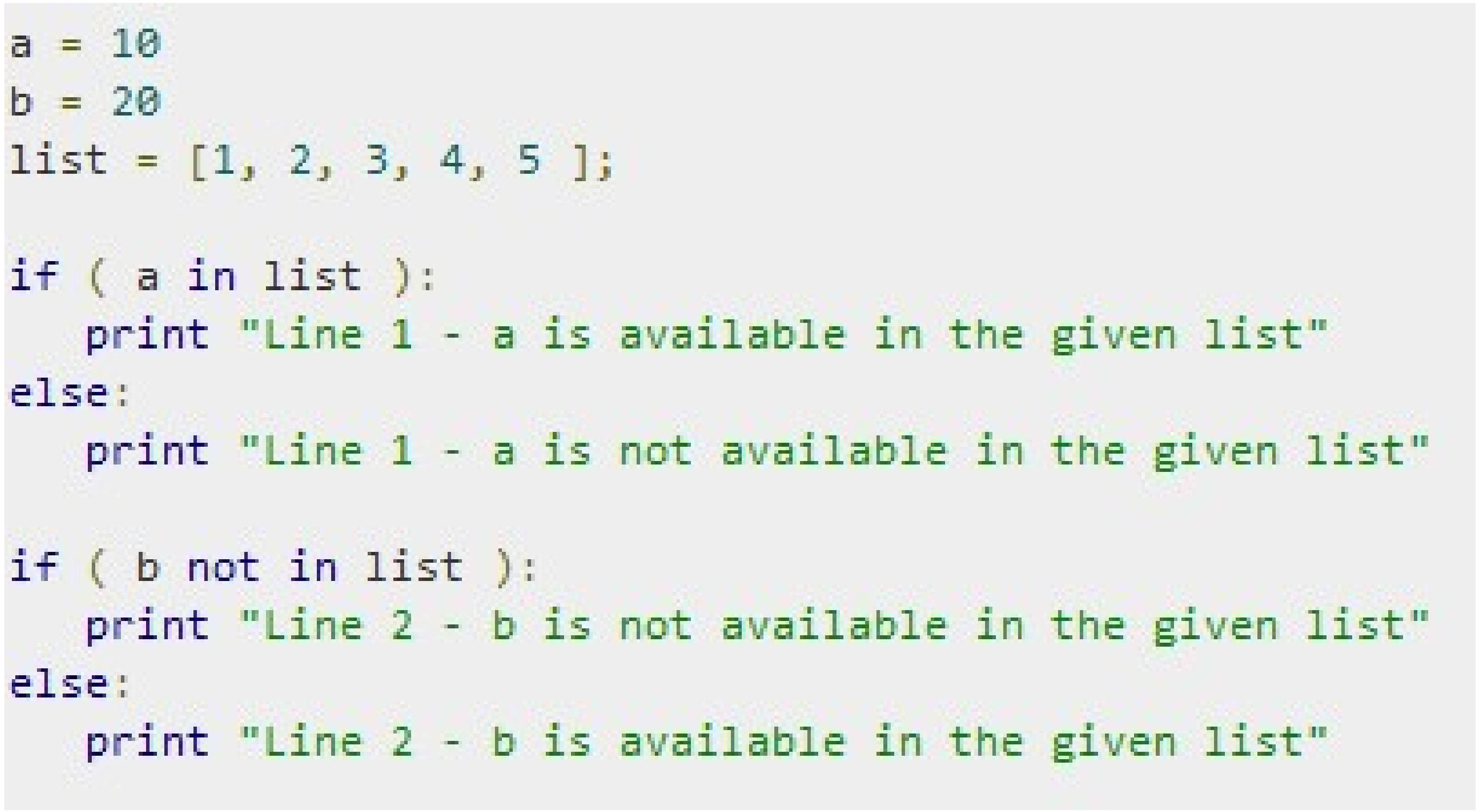
Python Membership Operators

- 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.

Python Membership Operators

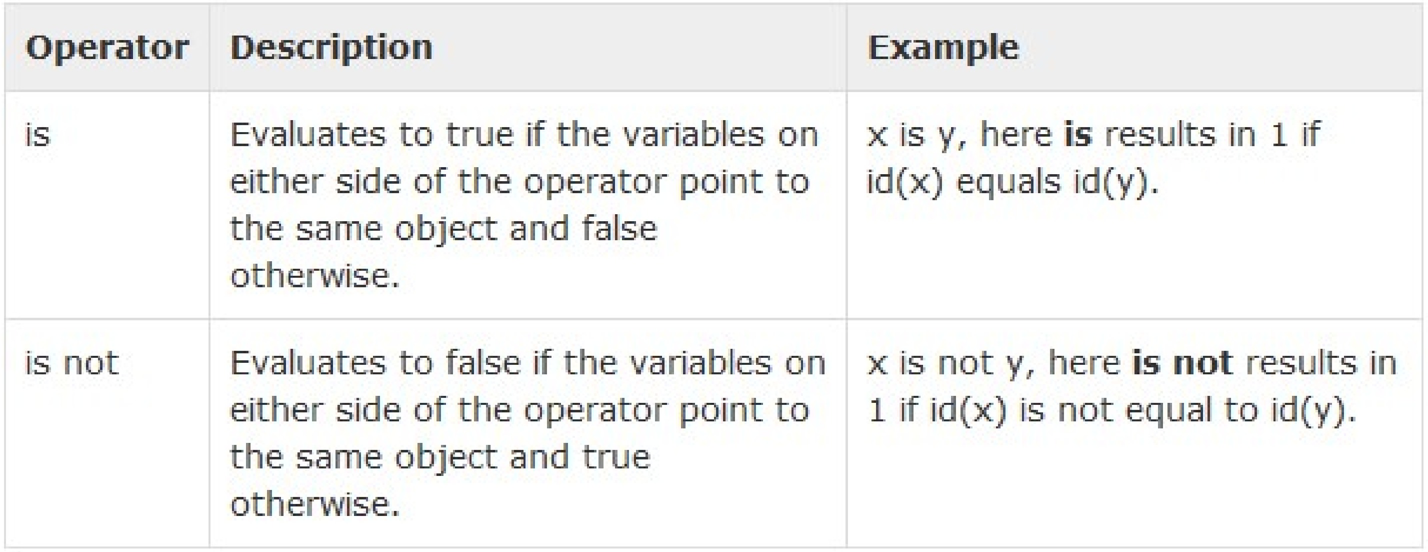


```python
a = 10
b = 20
list = [1, 2, 3, 4, 5 ];

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"
```

Note: print function should be written as print() from 3.x version of python.

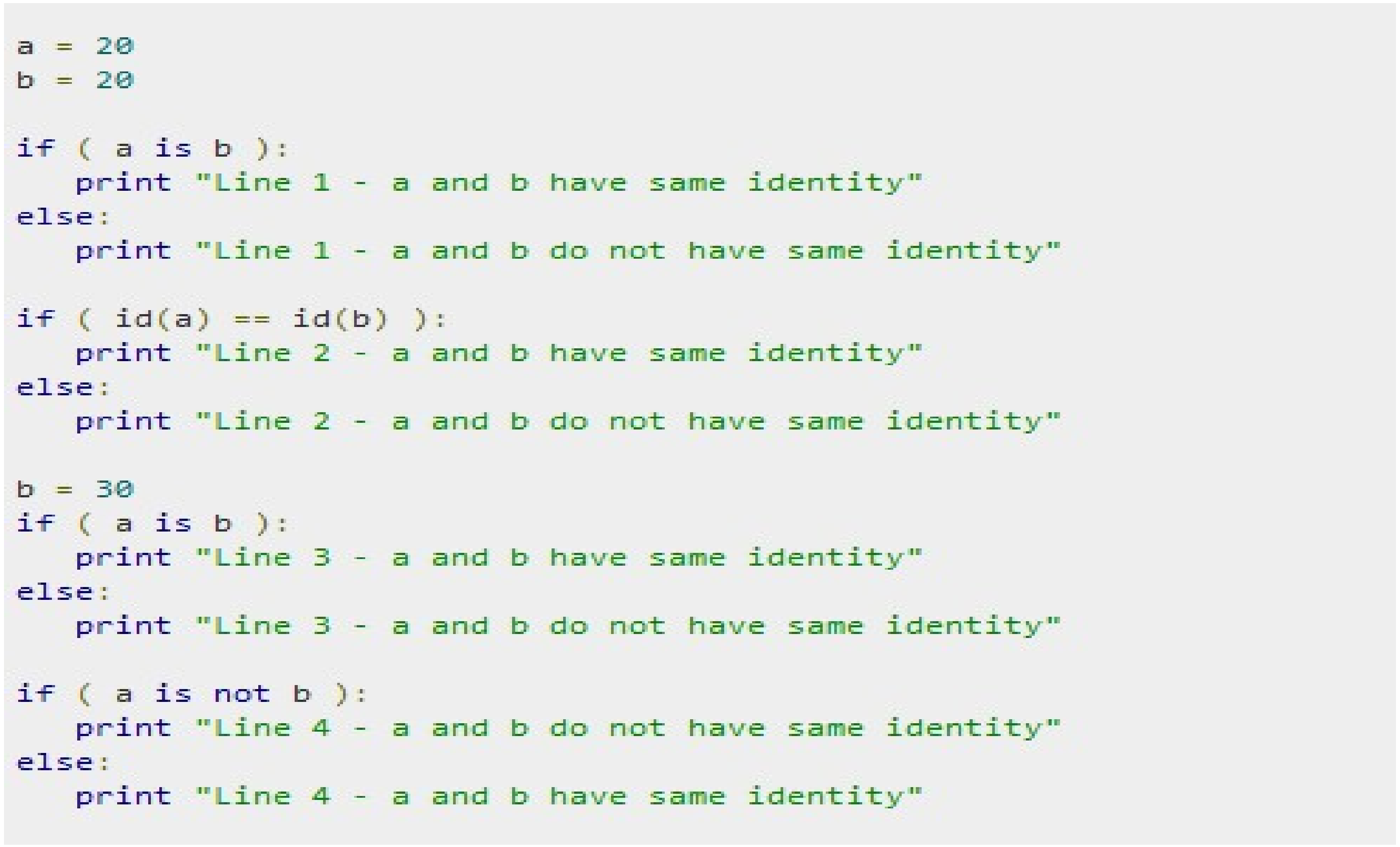
Python Identity Operators

- 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).

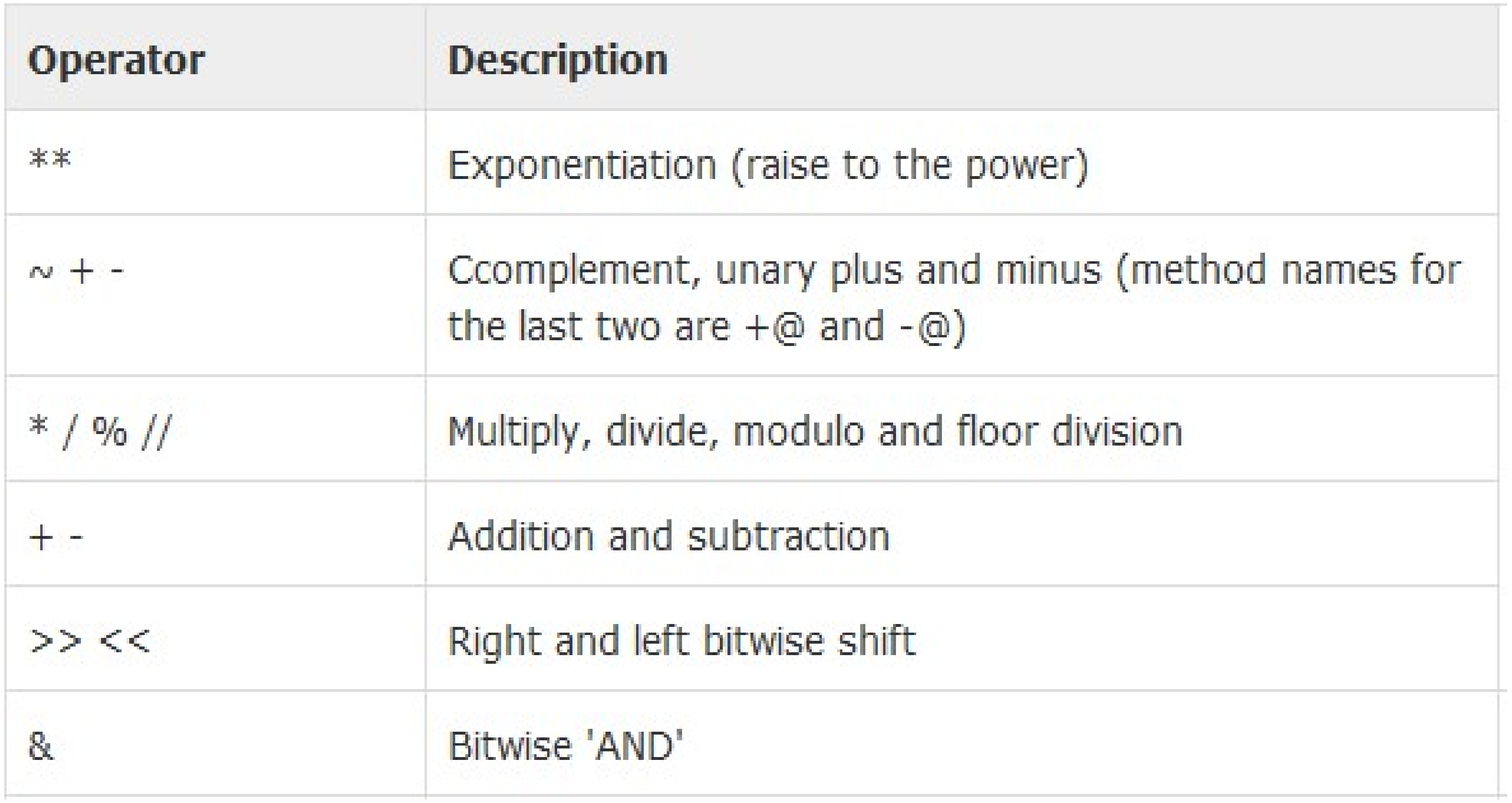
Python Identity Operators



```
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"
```

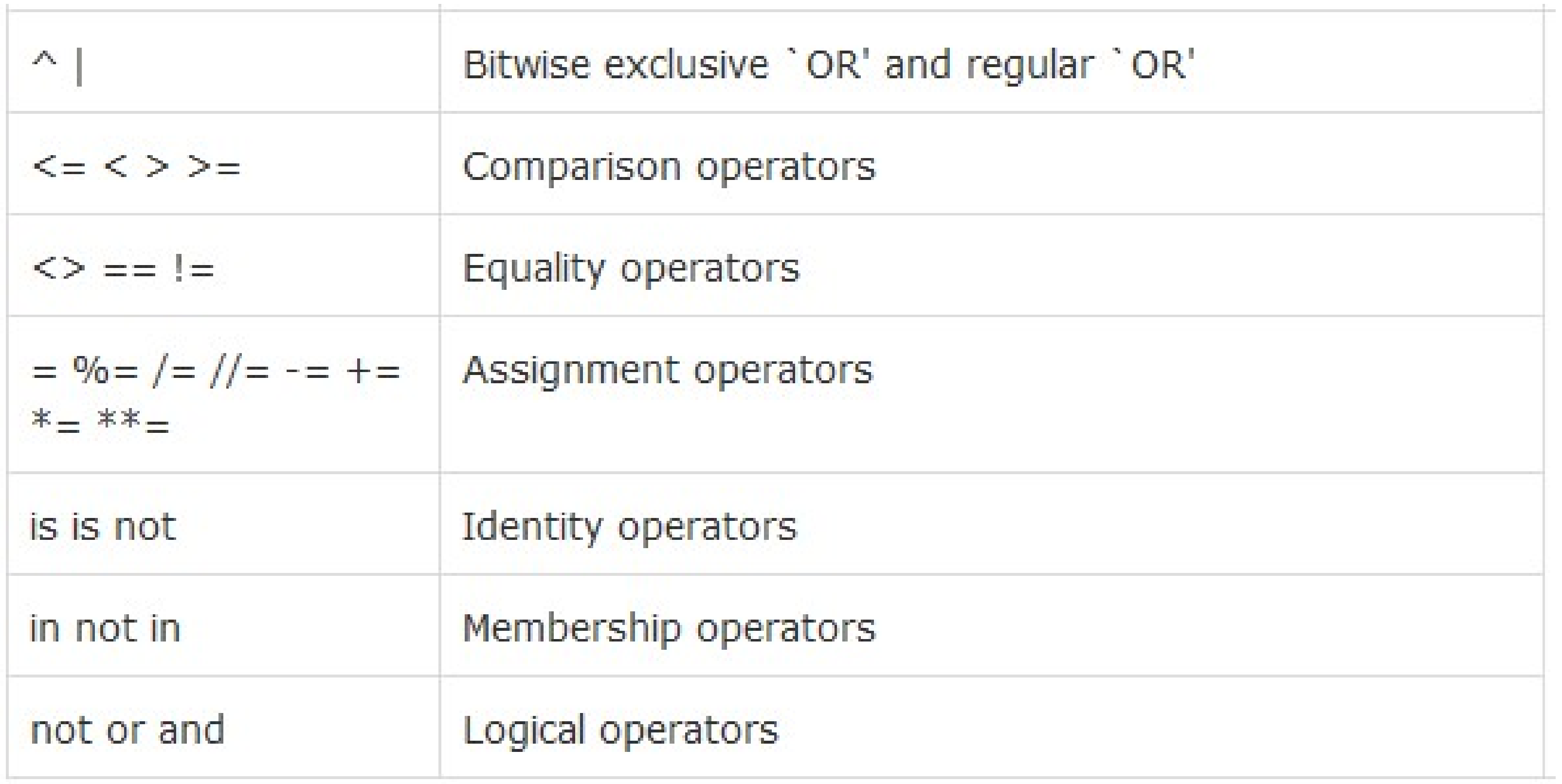
Note: print function should be written as print() from 3.x version of python.

Python Operators Precedence



Operator	Description
**	Exponentiation (raise to the power)
~ + -	Ccomplement, 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'

Python Operators Precedence



^ |	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

Using Python from the command line

```
$python

Python 3.3.2 (v3.3.2:d047928ae3f6, May 16 2013, 00:03:43) [MSC v.1600 32 bit (Intel)]
on win32

Type "copyright", "credits" or "license()" for more information.

>>>  print("2 + 2 is", 2+2)

        2 + 2 is 4

>>>print("3 * 4 is", 3 * 4)

        3 * 4 is 12

>>>print(100 - 1, " = 100 - 1“)

        99 = 100 – 1

>>>print("(33 + 2) / 5 + 11.5 = ",(33 + 2) / 5 + 11.5)

        (33 + 2) / 5 + 11.5 = 18.5
```

A small Python program

```
#!/usr/bin/python

print("2 + 2 is", 2+2)

print("3 * 4 is", 3 * 4)

print(100 - 1, " = 100 - 1“)

print( "(33 + 2) / 5 + 11.5 = ",(33 + 2) / 5 + 11.5)
```

The output when the program is run:

```
2 + 2 is 4

3 * 4 is 12

99 = 100 - 1

(33 + 2) / 5 + 11.5 = 18.5
```

Numbers

- The usual suspects
  - 12, 3.14, 0xFF, 0377, (-1+2)*3/4**5, abs(x), 0<x<=5

- C-style shifting & masking
  - 1<<16, x&0xff, x|1, ~x, x^y

- Integer division truncates :-(
  - 1/2 -> 0   # 1./2. -> 0.5, float(1)/2 -> 0.5
  - Will be fixed in the future

- Long (arbitrary precision), complex
  - 2L**100 -> 1267650600228229401496703205376L
  - 1j**2 -> (-1+0j)

precedence and associativity

- The order of operations is the same as in math:

1. parentheses ()
2. exponents **
3. multiplication *, division \, and remainder %
4. addition + and subtraction -

Strings

"hello"+"world"	"helloworld"	# concatenation
"hello"*3	"hellohellohello"	# repetition
"hello"[0]	"h"	# indexing
"hello"[-1]	"o"	# (from end)
"hello"[1:4]	"ell"	# slicing
len("hello")	5	# size
"hello" < "jello"	1	# comparison
"e" in "hello“	1	# search

- "escapes: \n etc, \033 etc, \if etc"
- 'single quotes' """triple quotes""" r"raw strings"

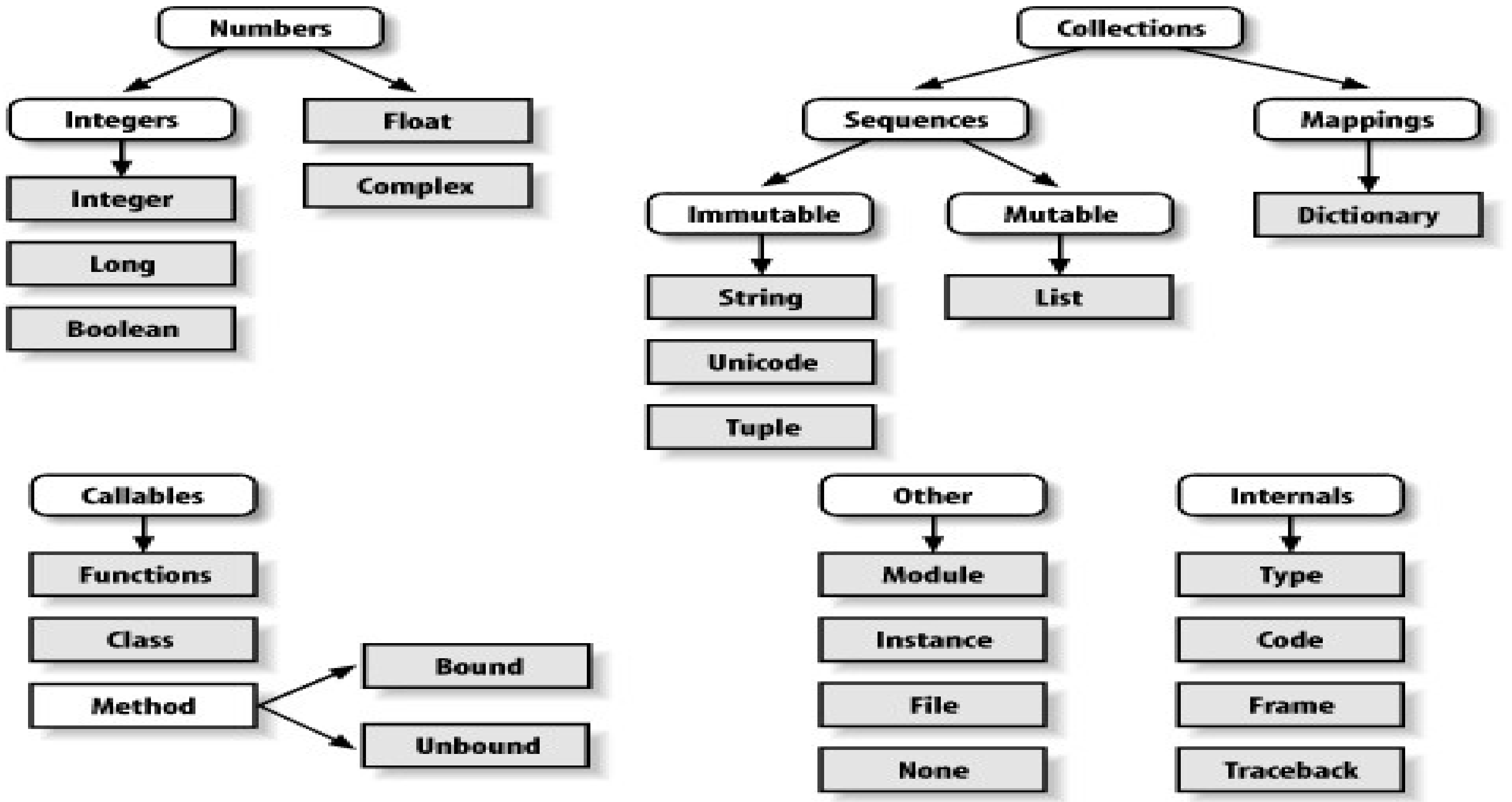
Note: print function should be written as print() from 3.x version of python.

Floating point numbers

- print "14 / 3 = ",14 / 3
- print "14 % 3 = ",14 % 3
- print
- print "14.0 / 3.0 =",14.0 / 3.0
- print "14.0 % 3.0 =",14 % 3.0
- print
- print "14.0 / 3 =",14.0 / 3
- print "14.0 % 3 =",14.0 % 3
- print
- print "14 / 3.0 =",14 / 3.0
- print "14 % 3.0 =",14 % 3.0

Note: print function should be written as print() from 3.x version of python.

Python’s Type Hierarchies



input/output operations

```
#!/usr/bin/python
#Author:
#Date:
#purpose : Input / Output and Variables
num = input("Type in a Number: ")
str = raw_input("Type in a String: ")
print "num =", num
print "num is a ",type(num)
print "num * 2 =",num*2
print "str =", str
print "str is a ",type(str)
print "str * 2 =",str*2
```

The output:

```
Type in a Number: 12.34
Type in a String: Hello
num = 12.34
num is a <type ’float’>
num * 2 = 24.68
str = Hello
str is a <type ’string’>
str * 2 = HelloHello
```

Note: print function should be written as print() from 3.x version of python.

Program Flow Constructs

Agenda

- control structures
- if Statements
- for Statements
- break and continue Statements
- else Clauses on Loops
- pass Statements
- The range() Function

Control Structures

```
if condition:
    statements
[elif condition:
    statements] ...
else:
    statements
```

```
while condition:
    statements

for var in sequence:
    statements

break
continue
```

if Statements

```
>>> x = int(input("Please enter an integer: "))
Please enter an integer: 42
>>> if( x < 0):
...     x = 0
...     print('Negative changed to zero')
... elif(x == 0):
...     print('Zero')
... elif(x == 1):
...     print('Single')
... else:
...     print('More')
...
More
```

TATA ELXSI

if Statements

```
#!/usr/bin/python
#Purpose :  Plays the guessing game higher or lower
```

```
number = 78
guess = 0
while guess != number :
    guess = input ("Guess a number: ")
    if(guess > number ):
        print("Too high“)
    elif(guess < number ):
        print("Too low“)

    print "Just right"
```

Note: print function should be written as print() from 3.x version of python.

TATA ELXSI

The for loop

- **The range() Function**

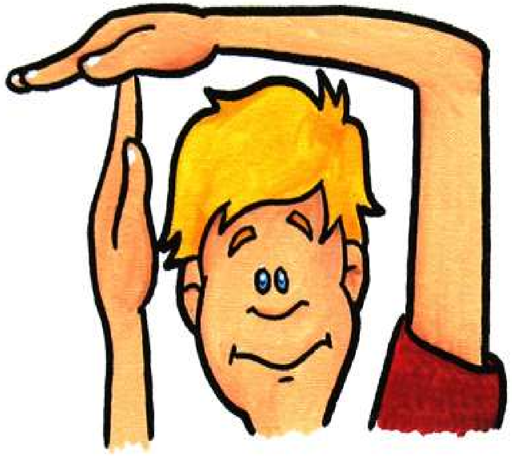
range(5, 10) 5 through 9
range(0, 10, 3) 0, 3, 6, 9
range(-10, -100, -30) -10, -40, -70

```
>>> for i in range(5):
 …    print(i)
...
```

Break statement

- The break statement, like in C, breaks out of the smallest enclosing for or while loop.

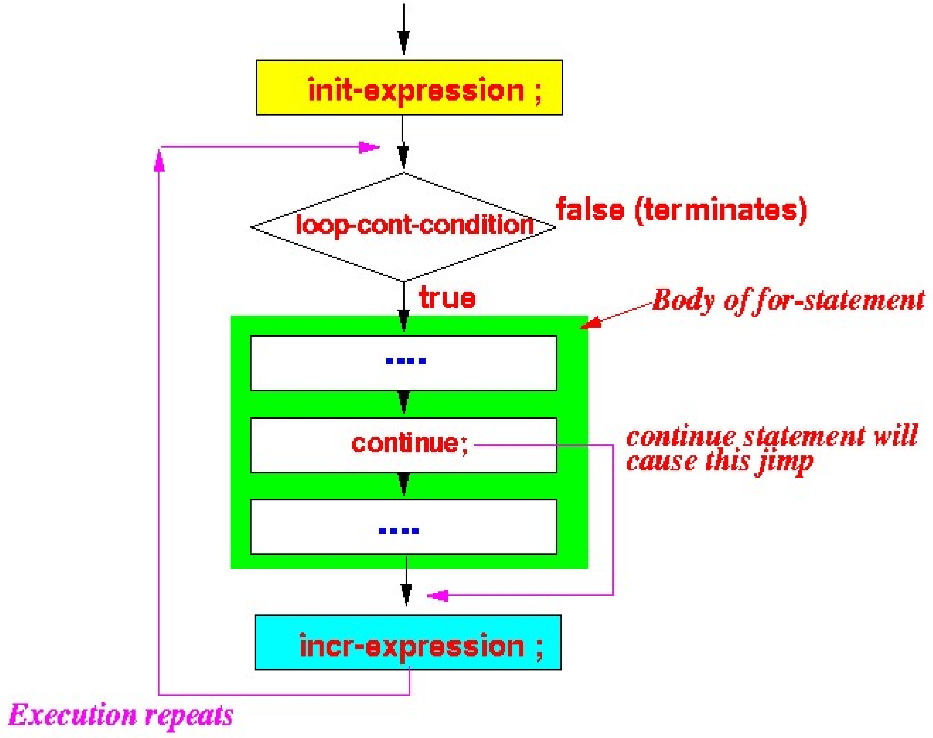
```
for n in range(2, 10):
...     for x in range(2, n):
...             if n % x == 0:
...                     print(n, 'equals', x, '*', n//x)
...                     break
...         else:
...             # loop fell through without finding a factor
...             print(n, 'is a prime number')
...
```



Continue statement

- The continue statement, also borrowed from C, continues with the next iteration of the loop:

```
>>> for num in range(2, 10):
...         if num % 2 == 0:
...                 print("Found an even number", num)
...                 continue
...         print("Found a number", num)
```



Else and loop

- Loop statements may have an else clause;
- It is executed when the loop terminates through exhaustion of the list (with for) or when the condition becomes false (with while).
- But not when the loop is terminated by a break statement.

```
>>> for n in range(2, 10):
...     for x in range(2, n):
...         if n % x == 0:
...             print(n, 'equals', x, '*', n//x)
...             break
...     else:
...       #loop fell through without finding a factor
...       print(n, 'is a prime number')
...
```

Pass Statements

- The pass statement does nothing.
- It can be used when a statement is required syntactically but the program requires no action.

```
>>> while True:
... pass # Busy-wait for keyboard interrupt (Ctrl+C) ...
```

This is commonly used for creating minimal classes:

```
>>> class MyEmptyClass:
... pass
...
```

Grouping Indentation

In Python:

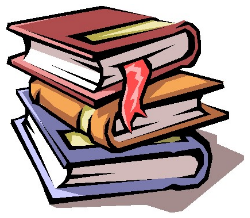
```
for i in range(20):
    if i%3 == 0:
        print(i)
        if i%5 == 0:
            print("Bingo!“)
    print("---“)
```

In C:

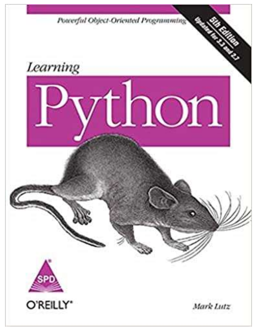
```
for (i = 0; i < 20; i++)
{
    if (i%3 == 0) {
        printf("%d\n", i);
        if (i%5 == 0) {
            printf("Bingo!\n"); }
      }
      printf("---\n");
}
```

```
0
Bingo!
---
---
---
3
---
---
---
6
---
---
---
9
---
---
---
12
---
---
---
15
Bingo!
---
---
---
18
---
---
```

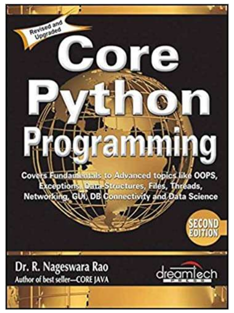
References



- **Python 3.x.x documentation**: https://docs.python.org/3/



- **Learning Python: Powerful Object-Oriented Programming: 5th Edition**



- **Core Python Programming [**by R. Nageswara Rao (Author)]

Thank you

Learning & Development

Tata Elxsi

Bangalore

www.tataelxsi.com

TATA ELXSI