Ground Rules!

- Mentally present – Observe!! Listen!!
- Keep your questions for the doubt session/ end of Session
- Solve the exercises regularly!
- Go that “extra mile” ☺

$$\boxed{1^{365} = 1}$$

$$\boxed{1.01^{365} = 37.8}$$

Motivation for this course ?

Motivation

- As of now, Python is one of the most widely used programming languages in the Data Science field.
- Data Scientists just love Python! ❤
- Python is easy to learn & has a great community for support!
- We would use Python for all the assignments / case-studies .

Course Objectives

What is this course about ?

- Introduce the fundamental programming concepts of Python
- Enable you to solve data problems using Python
- Act as a kick-start / bridge for participants of the programme who are ***new*** to Python.

What is this course *not* about ?

- Comprehensive, in-depth discussion about Python programming.
- Comprehensive, in-depth discussion about data analysis using Python and related packages, libraries, and tools.

DISCLAIMERS

- *The slides presented here are obtained from the authors of the books and from various other contributors. I hereby acknowledge all the contributors for their material and inputs.*
- *I have added and modified a few slides to suit the requirements of the course.*

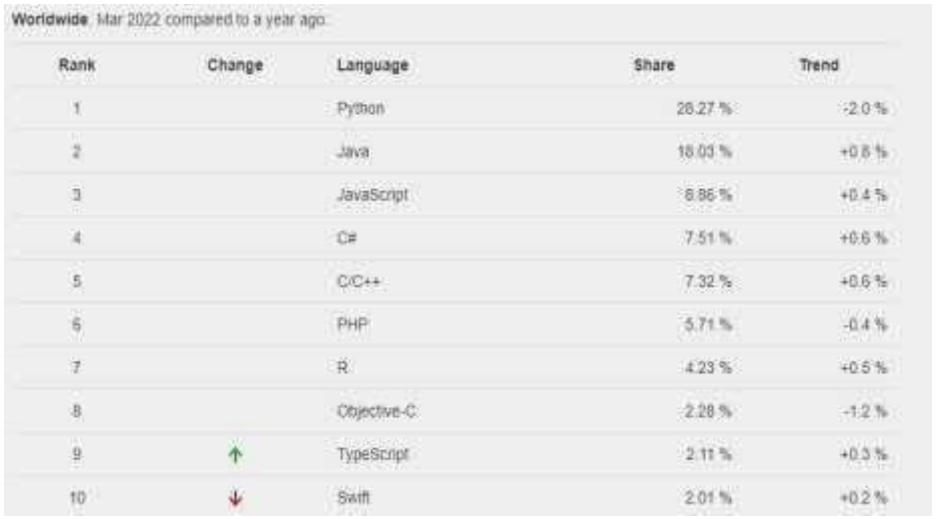
Program & Programming Language

Computer Program

- Set of instructions that perform a specific taskexecuted by computer
- Required by computer to function
- Written by programmer using programming languages
- Like C, C++, Java, Python etc.
- Executed with compiler and interpreter

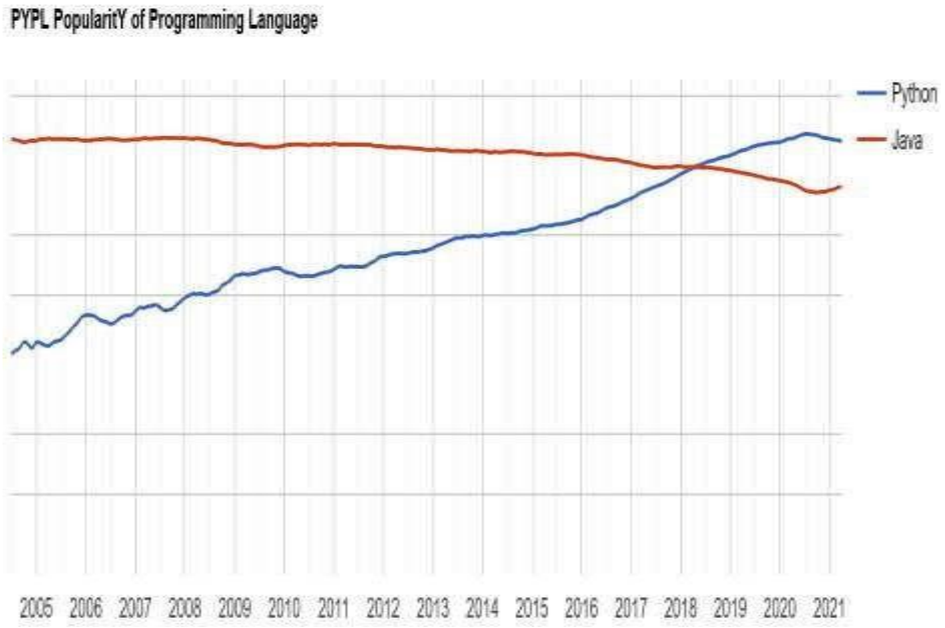
Python as aProgramming Language

Why Python ?



Worldwide, Mar 2022 compared to a year ago:

Rank	Change	Language	Share	Trend
1		Python	28.27 %	-2.0 %
2		Java	18.03 %	+0.8 %
3		JavaScript	8.86 %	+0.4 %
4		C#	7.51 %	+0.6 %
5		C/C++	7.32 %	+0.6 %
6		PHP	5.71 %	-0.4 %
7		R	4.23 %	+0.5 %
8		Objective-C	2.28 %	-1.2 %
9	↑	TypeScript	2.11 %	+0.3 %
10	↓	Swift	2.01 %	+0.2 %



Worldwide, Python is the most popular language … Source : http://pypl.github.io/PYPL.html

Python as aProgramming Language

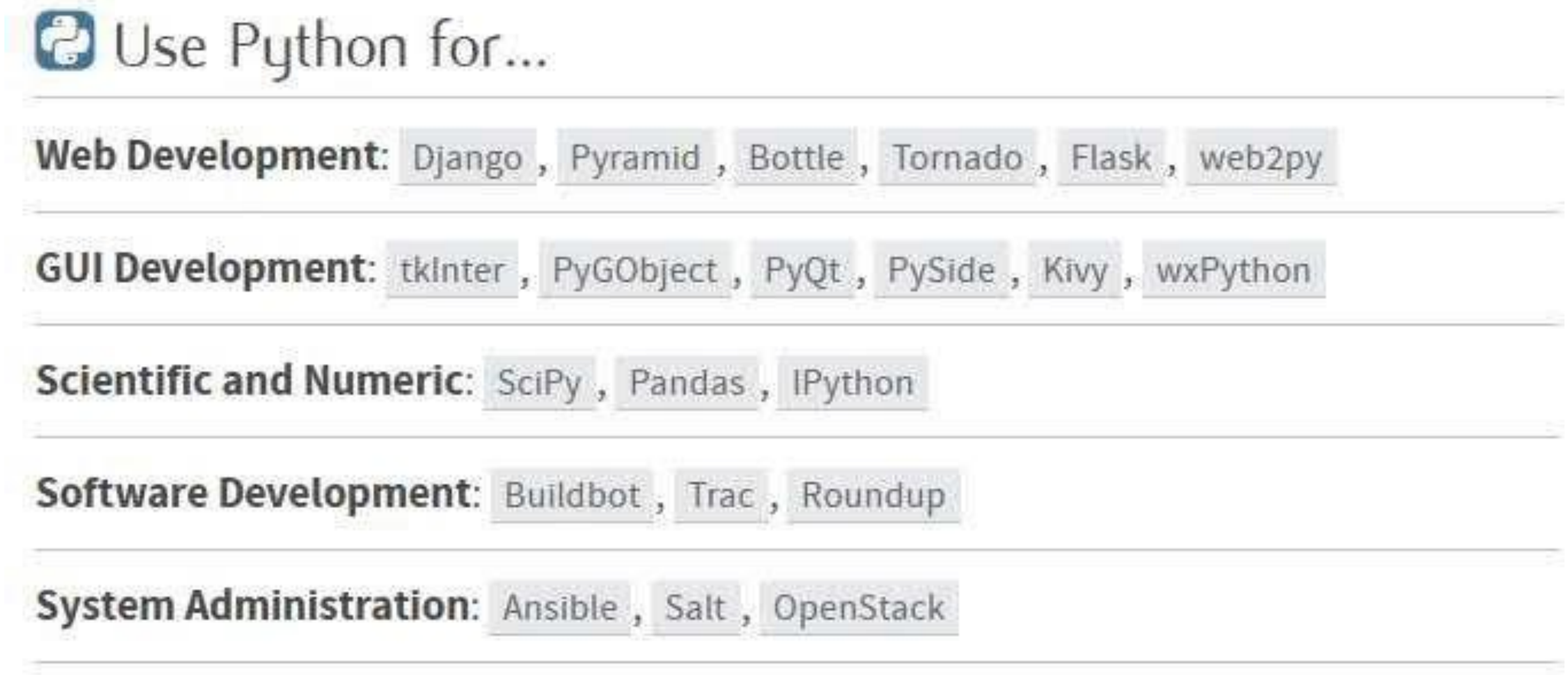
Python

- Designed by Guido van Rossum around 1990
- Not just a scripting language
- Easy to learn, read, use
- Extensible (add new modules)
- Highly readable
- Latest Version 3.9
- Most fond of language for Data Scientists

Touchy Feel Properties

- Open Source
  - copyrighted but use not restricted
  - owned by independentnon-profit, PSF
- Mature (29 years old)
- Supportive user community
  - plenty of good books, too
  - Active user community
- Simple design, easy to learn
  - reads like "pseudo-code"
  - Suitable as first language
  - Suitable as last language :-) (Hopefully)

Python Applications



Use Python for...

Web Development: Django, Pyramid, Bottle, Tornado, Flask, web2py

GUI Development: tkInter, PyGObject, PyQt, PySide, Kivy, wxPython

Scientific and Numeric: SciPy, Pandas, IPython

Software Development: Buildbot, Trac, Roundup

System Administration: Ansible, Salt, OpenStack

Python Ecosystem

Components of Python World :

- Core Python
- Distributions
- Frameworks / IDEs
- Third party Libraries

Core Python

- Programming Language itself
- Some standard modules are available
- Other packages needs to be explicitly installed

Python Distribution

- Python + packages
- Majority of packages, libraries are already available
- Package management is simplified
  - Anaconda from ContinuumAnalytics
  - IPython and its IPyKit variant

Python Ecosystem

Frameworks & IDEs

- Use frameworks to create code and develop applications
- Provides a defined structure to the developers so that they can focus on the core logic of the application rather than on other elements
- Python web framework
  - ✓ Django
  - ✓ Web2py
  - ✓ Flask
- Python IDEs
  - ✓ IDLE
  - ✓ PyCharm
  - ✓ Spyder
  - ✓ JupyterNotebooks

Third party Libraries

- Makes life of developers very simple
- Just need to know the right library to carry out a task
  - NumPy
  - Scipy
  - Pandas
  - Matplotlib
  - Seaborn
  - Bokeh
  - ScikitLearn
  - And List goes on …

Python Installation

Three Ways :

- Install Python directly
  - Install the Python language with installer
  - Need to install other packages explicitly using pip install
  - https://www.python.org/downloads/
- Use Python distribution
  - The open-source Anaconda Distribution is the easiest way to perform Python coding
  - Works on Linux, Windows, and Mac OS X
  - https://docs.anaconda.com/anaconda/install/windows/
- Use Cloud based services
  - The simplest of all but needs internet connectivity to use
  - Microsoft Azure Notebooks
  - Google Collab

Integrated Development Environments (IDE) forPython

Common IDE’s:

Our Favourite (For DSE):

○**.py** is a regular python file. It's plain text and contains just your code.
○**.ipynb** is a python notebook and it contains the notebook code, the execution results and other internal settings in a specific format.

Input / Output with Python

- print() can be used to output a message
- input() can be used to enter an input to the python program.
- # can be used to provide comments.
- ““““““(triple quotes) can be used to write documentation.

Demo:

- Let’s see how to launch Jupyter Notebook
- See the basics of Notebook
- Practice some I/O statements and comments.

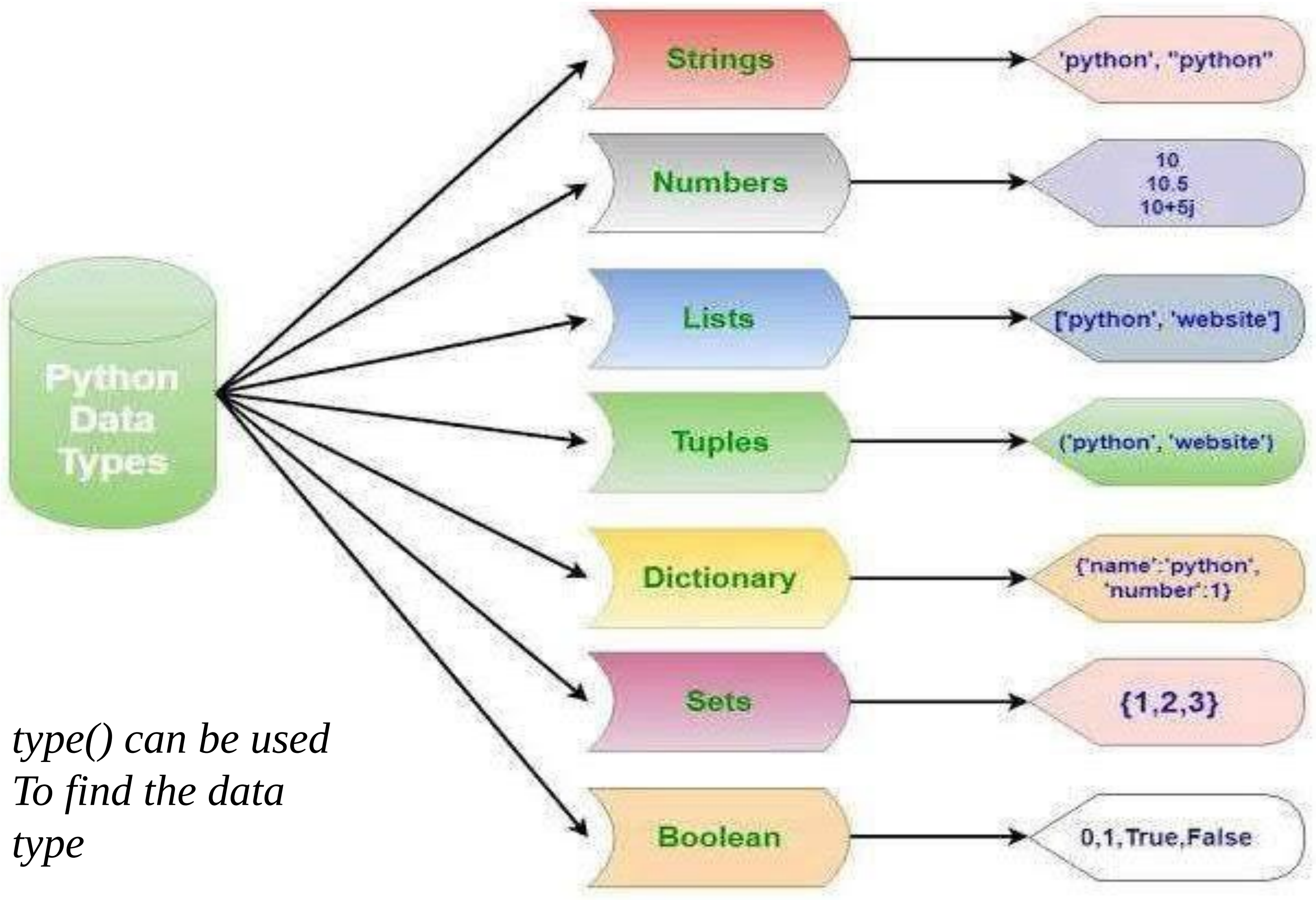
Basic Code Constructs

Imports:

- Import in Python is similar to #include in C/C++. Python modules can get access to code from another module by importing the file/function using import.
- Ex: import math
- print(math.pi) Variable

- A Python variable is a reserved memory location to store values. In other words, variables are containers for storing data values.
- *Python has no command for declaring a variable.*
- A variable is created the moment you first assign a value to it.
- Ex: a = 100

Data Types in Python



Immutable

✓

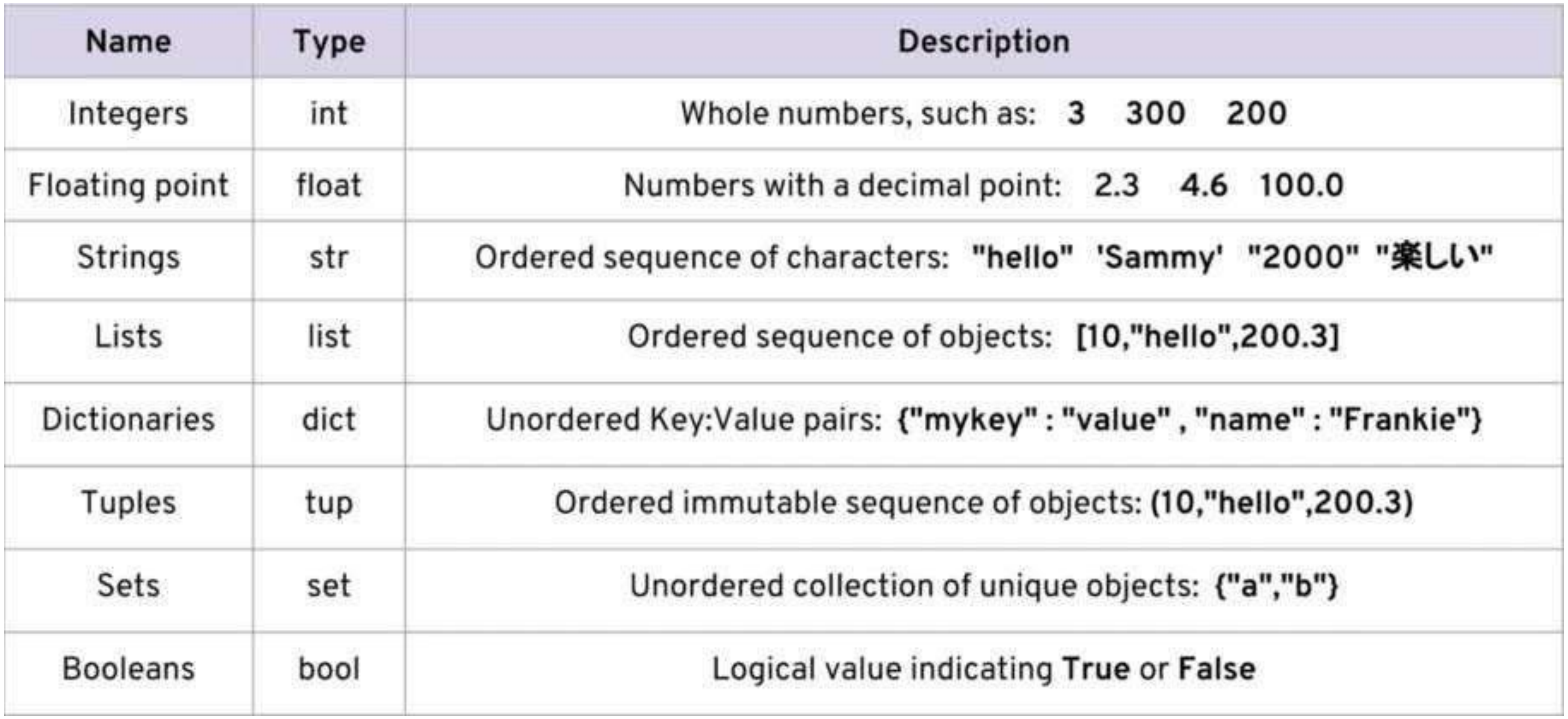
✓

✓

✓

type() can be used To find the data type

Data Types in Python



Name	Type	Description
Integers	int	Whole numbers, such as: **3 300 200**
Floating point	float	Numbers with a decimal point: **2.3 4.6 100.0**
Strings	str	Ordered sequence of characters: **"hello" 'Sammy' "2000" "楽しい"**
Lists	list	Ordered sequence of objects: **[10,"hello",200.3]**
Dictionaries	dict	Unordered Key:Value pairs: **{"mykey" : "value" , "name" : "Frankie"}**
Tuples	tup	Ordered immutable sequence of objects: **(10,"hello",200.3)**
Sets	set	Unordered collection of unique objects: **{"a","b"}**
Booleans	bool	Logical value indicating **True** or **False**

type() can be used to find the data type

Post your queries in the Discussion Forum!!

Agenda for CS #2

1) Immutable & Mutable Data Structures
2) String & its Operations
3) Tuple & its Operations
4) List & its Operations
5) Set & its Operations
6) Dictionary & its Operations
7) Operators
8) Conditional Statements
9) Exception Handling

Immutable & Mutable Objects

Mutable Objects:

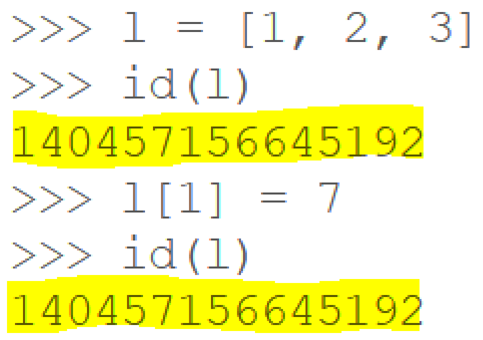
Some objects are mutable, meaning their values can change throughout the execution of a program. In practice, this means they maintain the same id even after a change in value. This is known as an ‘in-place’ change because it doesn’t require the creation of a new object to change in value. Mutable objects include lists, dictionaries, and sets.

Immutable objects:

Immutable objects on the other hand can not change in value during the execution of a program. The only way to alter the value of a variable that points to an immutable object, is to create an entirely new object containing the altered value. In other words, immutable objects can not be changed ‘in-place.’ Numbers, String, tuples are examples.

Immutable & Mutable

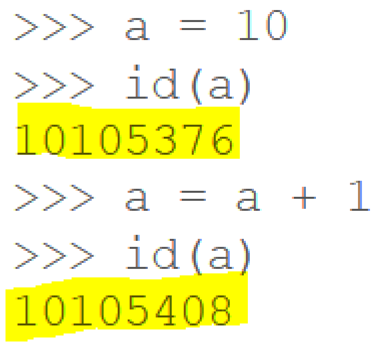
Mutable Structures



```
>>> l = [1, 2, 3]
>>> id(l)
140457156645192
>>> l[1] = 7
>>> id(l)
140457156645192
```

As you can see above, even though the contents of List l is modified, it’s id remains the same. Mutable objects in python include lists, sets, dictionaries.

Immutable Structures



```
>>> a = 10
>>> id(a)
10105376
>>> a = a + 1
>>> id(a)
10105408
```

As you can see above, once the value of integer a is modified, so is it’s id, because it now points to a different object and a different space in memory. Immutable objects in python include numbers (integers and floats), strings, bools, tuples, and more.

Demo on Python Data Types / Structures

Lets have a hands-on session on :

- String & its operations
- Tuple & its operations
- List & its operations
- Set & its operations
- Dictionary & its operations

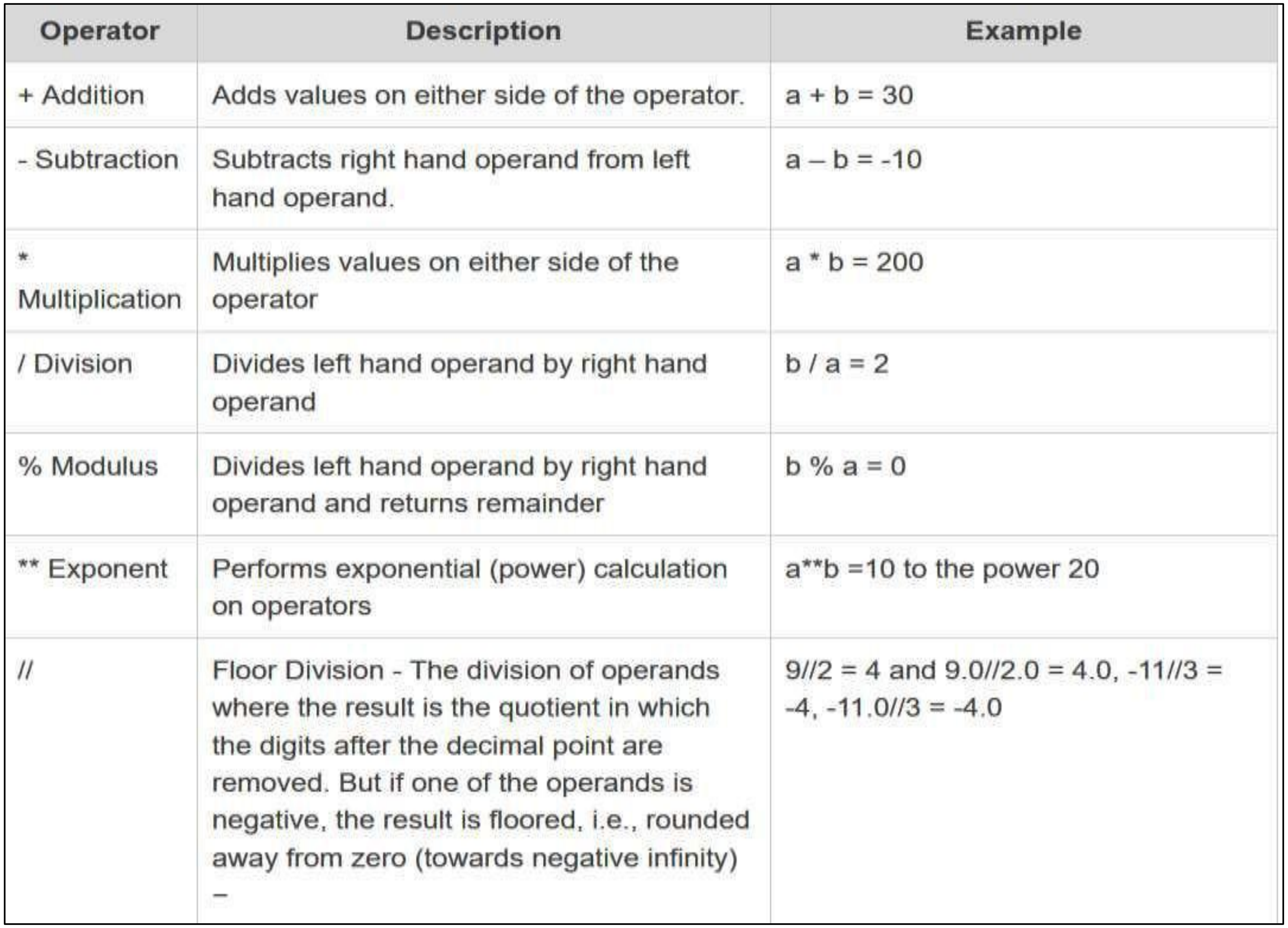
Operators in Python

Python language supports the following types of operators:

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

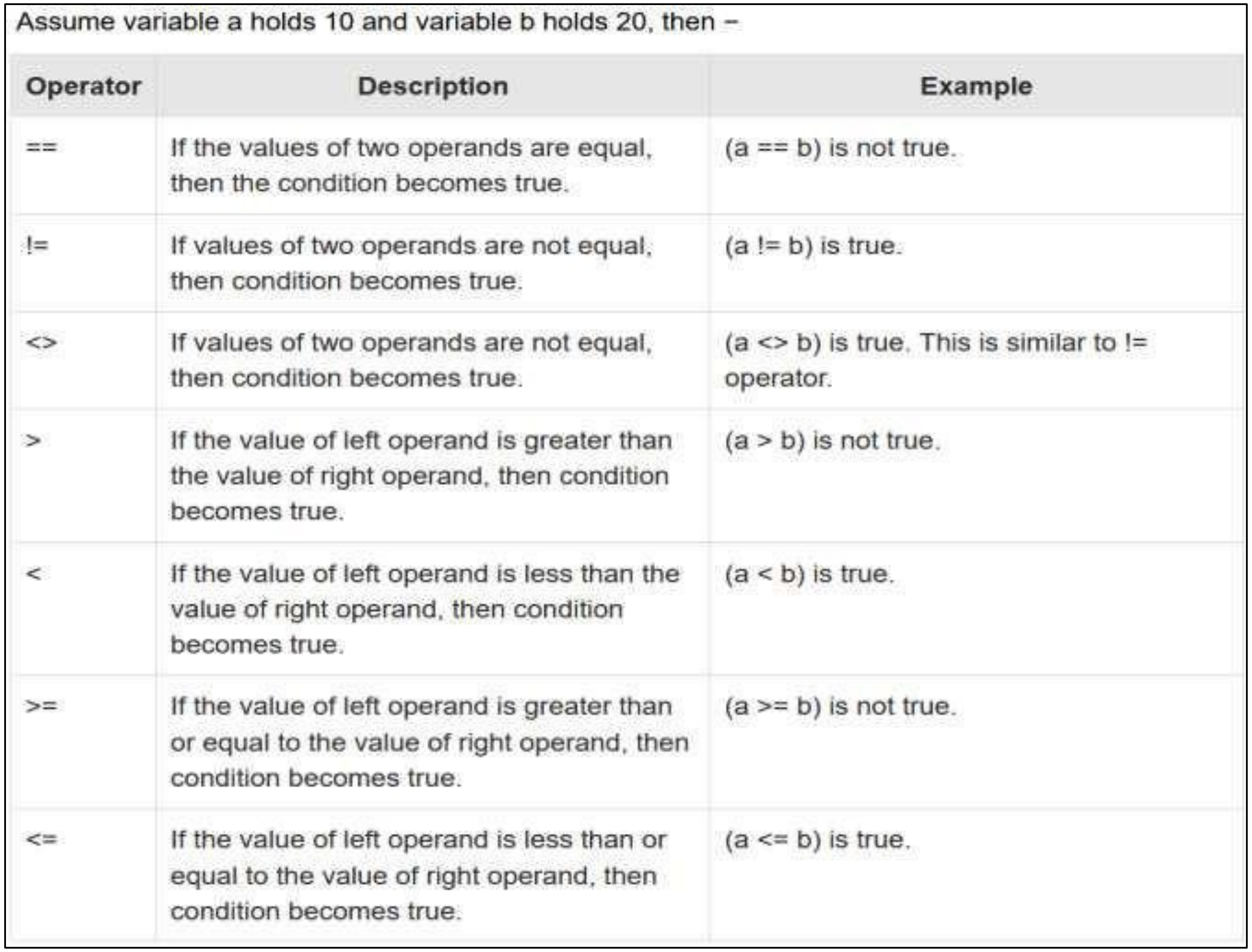
Arithmetic Operators

Assume
a=10
b=20



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

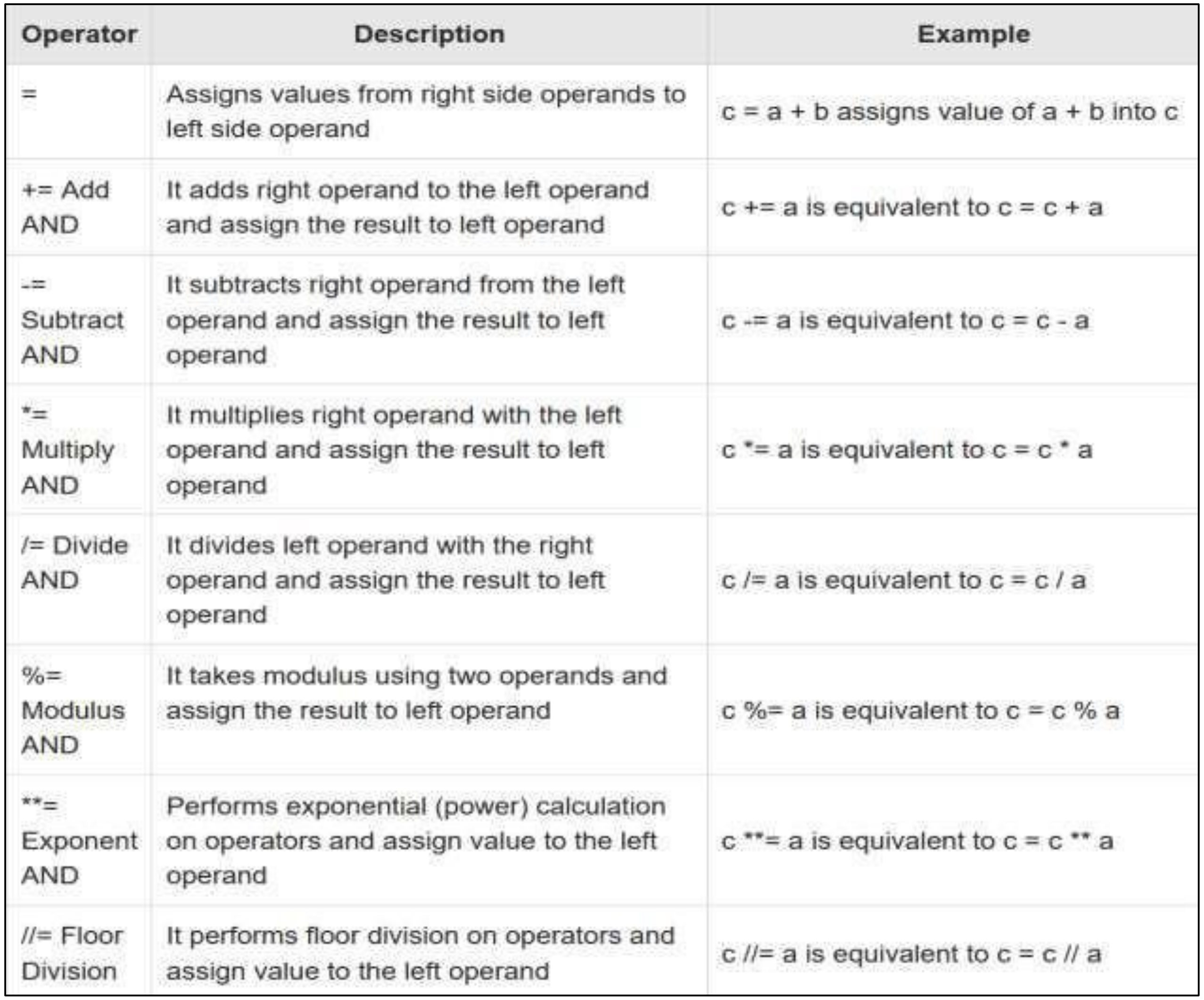
Comparison (Relational) Operators



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

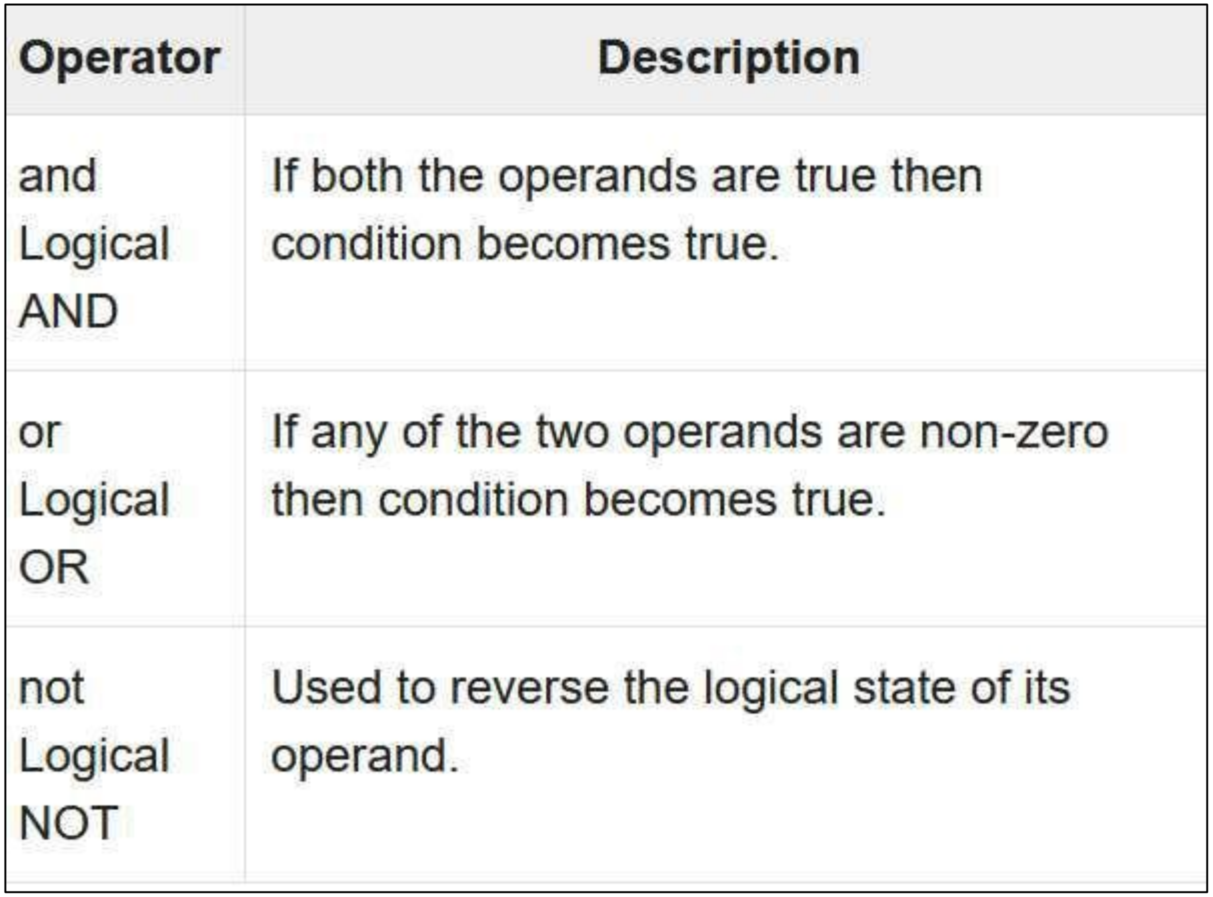
Operator	Description	Example
==	If the values of two operands are equal, then the condition becomes true.	(a == b) is not true.
!=	If values of two operands are not equal, then condition becomes true.	(a != b) is true.
<>	If values of two operands are not equal, then condition becomes true.	(a <> b) 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.

Assignment Operators



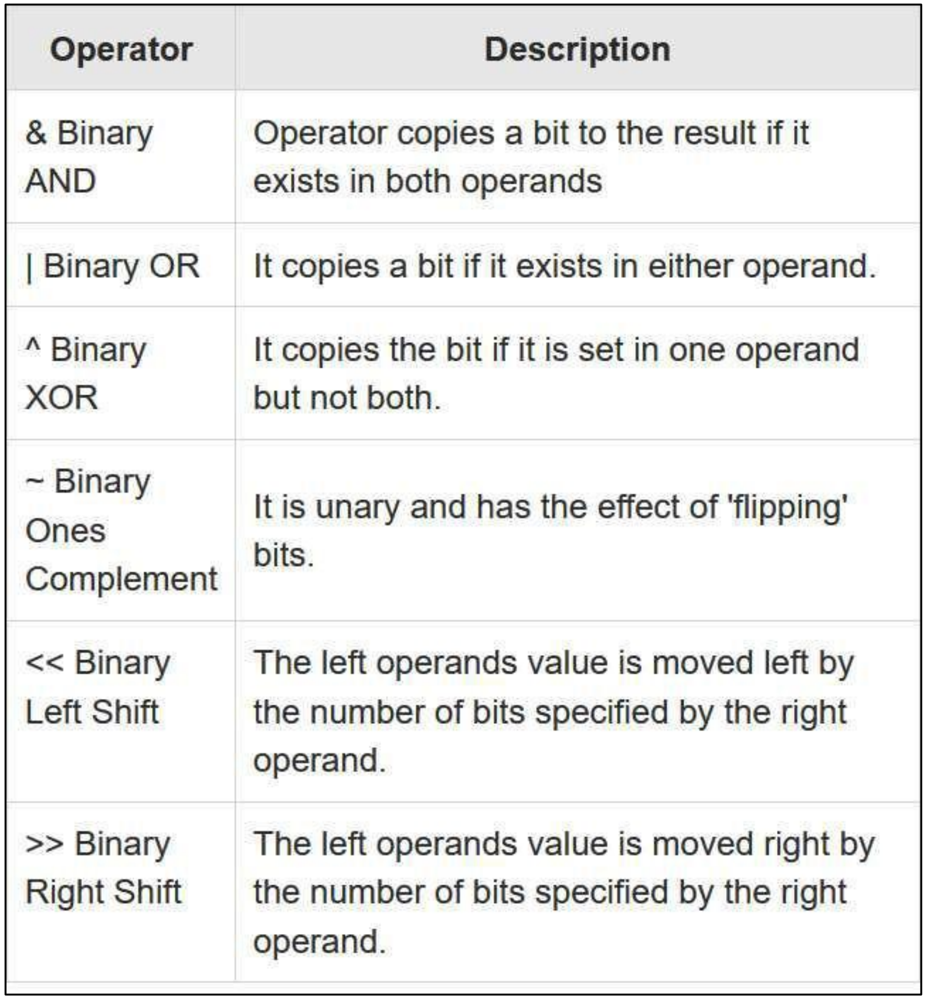
Operator	Description	Example
=	Assigns values from right side operands to left side operand	c = a + b assigns value of a + b into c
+= Add AND	It adds right operand to the left operand and assign the result to left operand	c += a is equivalent to c = c + a
-= Subtract AND	It subtracts right operand from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

Logical Operators



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

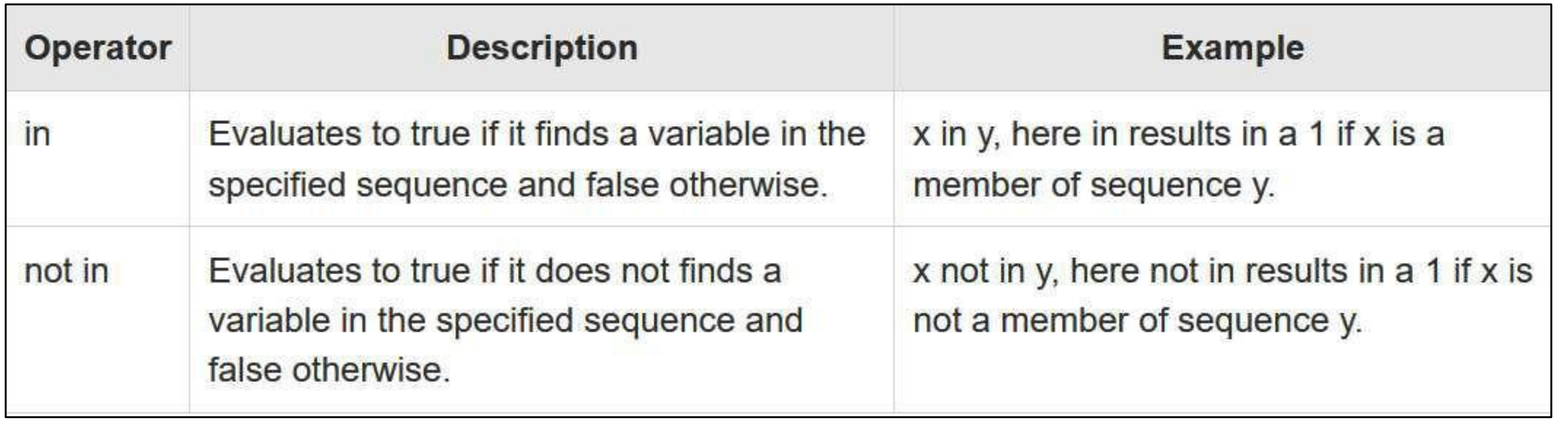
Bitwise Operators



Operator	Description
& Binary AND	Operator copies a bit to the result if it exists in both operands
| Binary OR	It copies a bit if it exists in either operand.
^ Binary XOR	It copies the bit if it is set in one operand but not both.
~ Binary Ones Complement	It is unary and has the effect of 'flipping' bits.
<< Binary Left Shift	The left operands value is moved left by the number of bits specified by the right operand.
>> Binary Right Shift	The left operands value is moved right by the number of bits specified by the right operand.

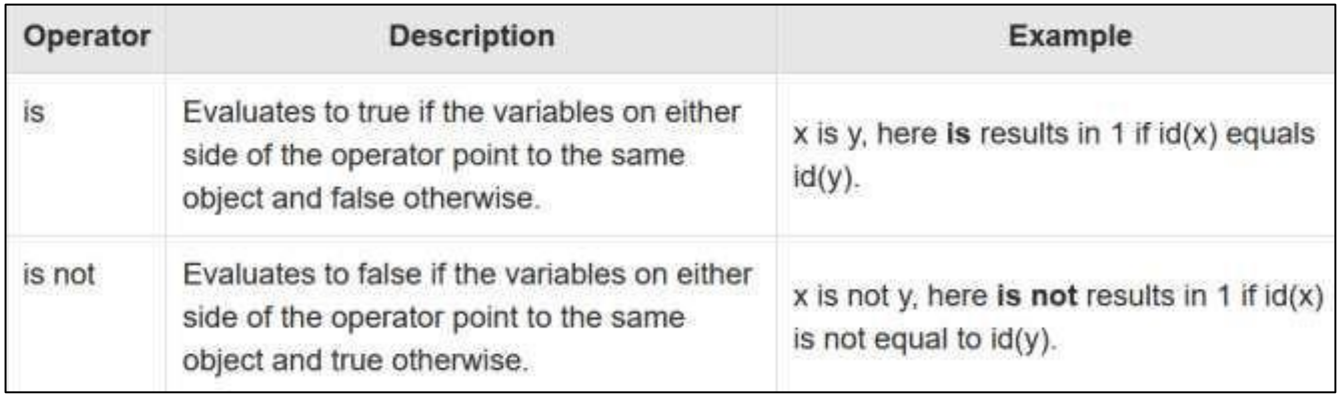
Membership & Identity Operators

Python’s membership operators test for membership in a sequence, such as strings, lists, or tuples:



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.

Identity operators compare the memory locations of two objects.

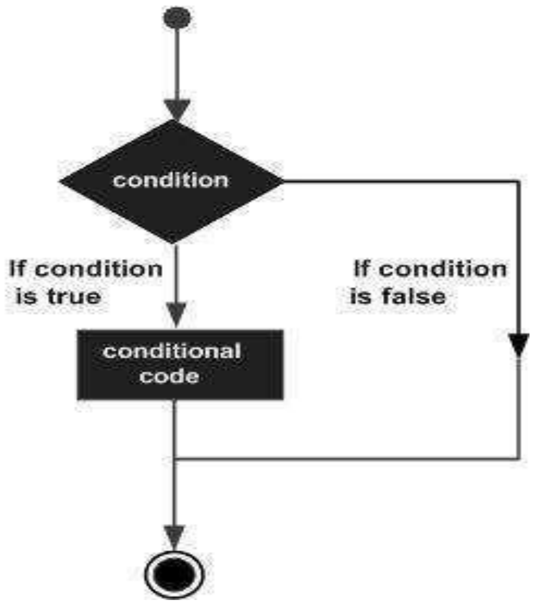


Operator	Description	Example
is	Evaluates to true if the variables on either side of the operator point to the same object and false otherwise.	x is y, here **is** results in 1 if id(x) equals id(y).
is not	Evaluates to false if the variables on either side of the operator point to the same object and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

Conditional Execution in Python (Decision Making)

- Decision making is anticipation of conditions occurring while execution of the program and specifying actions taken according to the conditions.
- Decision structures evaluate multiple expressions which produce TRUE or FALSE as outcome. You need to determine which action to take and which statements to execute if outcome is TRUE or FALSE otherwise.
- Following is the general form of a typical decision making structure found in most of the programming languages −

*Python programming language assumes any **non-zero** and **non-null** values as TRUE, and if it is either **zero** or **null**, then it is assumed as FALSE value.*



Decision Making in Python

Python Supports the following:

- If statements
- If … else statements
- Nested If statements
- Chained If statements

```
if expression:
    statement(s)
Statement after if condition
```

```
if expression:
statement(s) else:
statement(s)
Statement    after  if
    condition
```

```
if expression1:
    statement(s)
else:
     if expression2:
        statement(s)
     else:
        statement(s)
```

```
if expression1:statement(s)
elif  expression2:
    statement(s)
else:
statement(s)
```

Exception Handling

The Exception Handling in Python is one of the powerful *mechanism to handle the runtime errors* so that normal flow of the application can be maintained and user friendly error messages can be displayed.

- The **try** block lets us test a block of code for errors.
- The **except** block lets us handle the error.
- The **finally** block lets us execute code, regardless of the result of the try and except blocks.

```
try:
 a = 1/0
except Exception as e:
 print(e)
finally:
 print(“I am always there”)
```

Feedback

🙂 👍 : 5

🤨 🤞 : 3

😒 👎 : 1

Thank You for your time & attention !