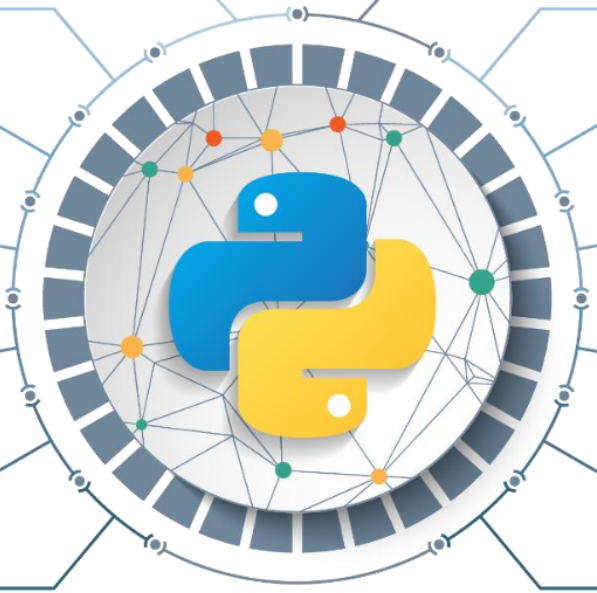
Course Outline



1. Introduction to Python
2. Sequences and File Operations
3. Deep Dive-Functions, OOPS, Modules, Errors and Exceptions
4. Introduction to Numpy, Pandas and Matplotlib
5. Data Manipulation
6. Introduction to Machine Learning with Python
7. Supervised Learning - I
8. Dimensionality Reduction
9. Supervised Learning - II
10. Unsupervised Learning
11. Association Rules Mining and Recommendation Systems
12. Reinforcement Learning
13. Time Series Analysis
14. Model Selection and Boosting

edureka!

Introduction to Python

Topics

The topics covered in this module are:

- Need of Programming
- Companies using Python
- Applications where Python is Used
- Python Interpreter
- Values, Types, Variables
- Operands and Expressions
- Write your First Python Program
- Command Line Arguments

Objectives

After completing this module, you should be able to:

- Define Python
- Understand why Python is Popular
- Know where Python is used in Applications
- Setup Python Environment
- Discuss Python Interpreter
- Define Values, Types, Variables
- Understand Operands and Expressions
- Write your First Python Program
- Work with Conditional Statements and Loops
- Understand Command Line Parameters and Flow Control

Need of Programming

edureka!

Course Management System

John's job is to film and edit courses for a website

John

Course Management System

John films and edits courses for a website

John

Course Management System

In the process of preparing a course, John needs to deal with many different files

Video Files

Audio Files

Motion Graphics

He creates a set of folders to organize these materials by course, lesson and type of file

Physics 

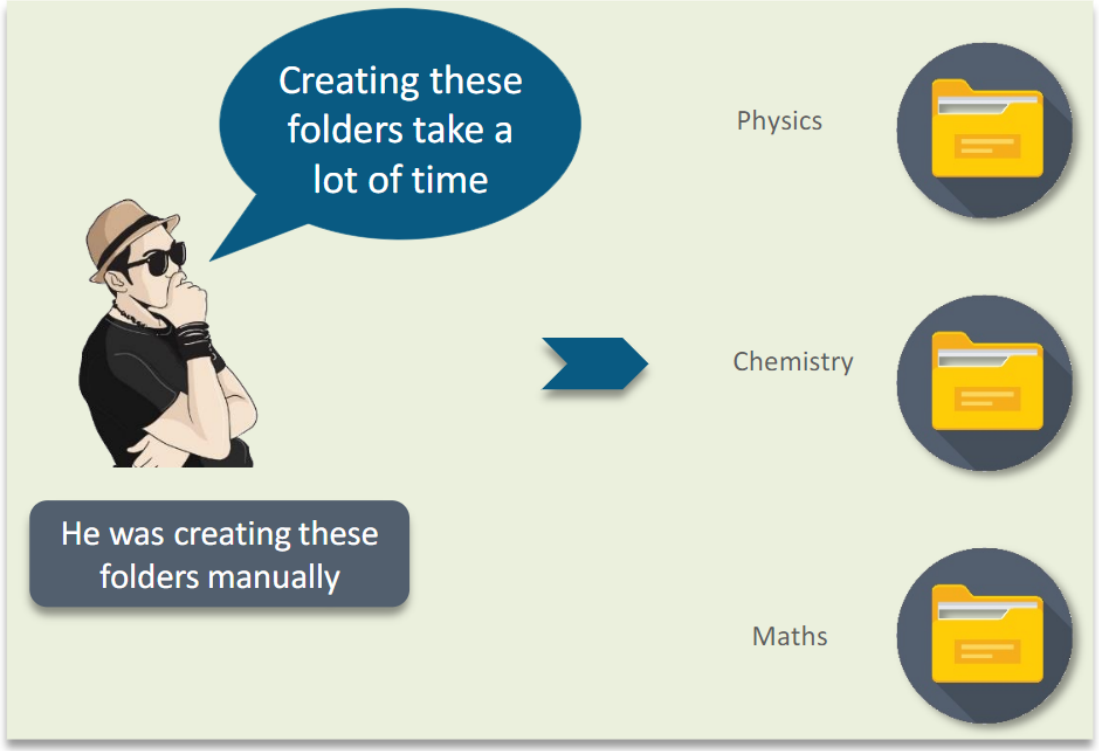
Chemistry 

Maths 

edureka!

Course Management System – Problem Statement

John wants something to automate this task, and save a lot of time



Hospital Management System

Annie is a receptionist in a hospital

Annie

Hospital Management System – Problem Statement

She gets annoyed because, she has to give hard copy of the report to the patients

Hospital Management System

She wants to create a system which will link Patient's report to their given Aadhar Card Number

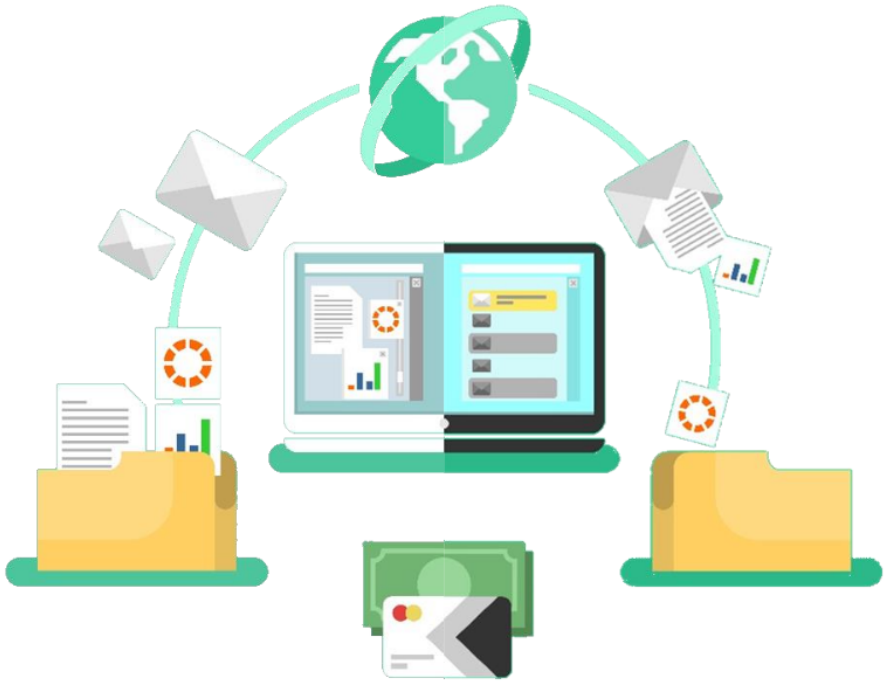
Annie

edureka!

Solution - 1

John learnt programming and designed a system, which automatically creates folders, name of the course, number of lessons at the end of the course

John

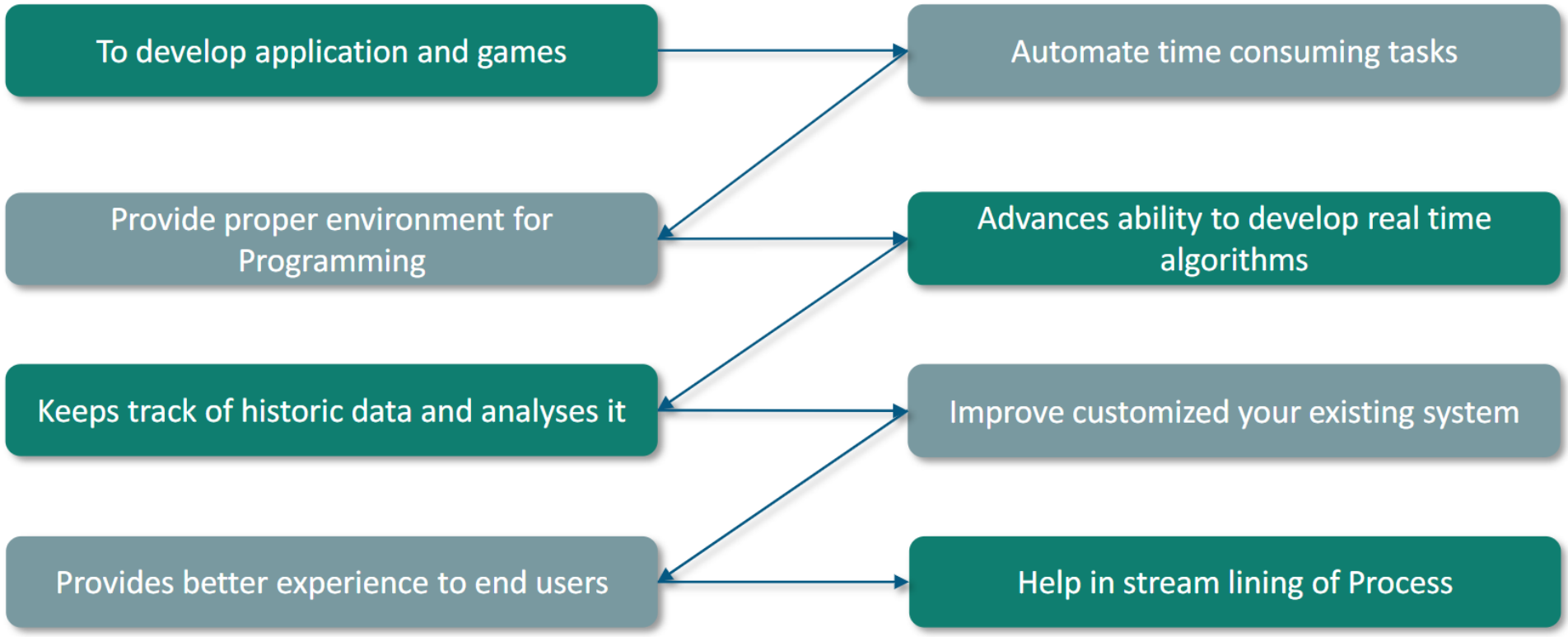


Solution - 2

Annie wrote a *Program* to design a system, in which a patient has to enter his/her Aadhar Card Number and reports will be linked to his number

Annie

Need of Programming



Advantages of Programming

With Out Computer Programs	With Computer Programs
Communicate with others: 3-4 days for sending a letter to some one even in the same district and it costs money	Less than 1-minute to send an email to any place in the world at no cost
Booking Railway Tickets: Stand in really long queues. Can reserve railway tickets only for the quota of that station. Even if other stations quota is not filled, we can not reserve the tickets. Can only book tickets from that station and can not book return tickets	Can book tickets from anywhere to anywhere, sitting at home
Sending money from parent to the child studying in a college: This involves the parent going to the bank for the DD, send it to the student by post, student submitting the DD in the bank, wait for the DD to clear and withdraw the money by showing the passbook and filling an withdrawal form	The money transfer is almost instant and money can be withdrawn from any ATM
Seat allocation in engineering college: Go to the counseling center far away from your town, wait for hours together for your turn, but make the decision of which college and branch to choose in less than 5 minutes	Have lots of time to research the colleges, decide upon the colleges and branches, review them multiple people and submit it online with peace of mind

Why Python?

edureka!

Different Programming Languages

C++

Matlab

HTML

Python

Scala

Java

PHP

Dave tell me which Programming language should I start with

It has to be Python, let me give you enough reasons to believe that

edureka!

Why Python?

Simple and Easy to Learn

Python is simple and easy to learn, read & write

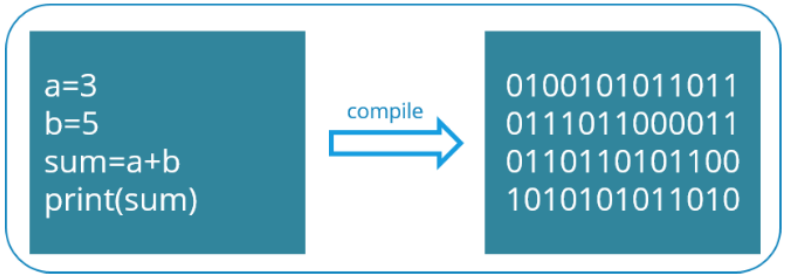
Free and Open Source

Python is an example of a FLOSS (Free/Libre and Open Source Software) which means one can freely distribute copies of this software, read it's source code, modify it, etc.

Why Python?

High-level Language

One does not need to bother about the low-level details like memory allocation, etc. while writing a Python script



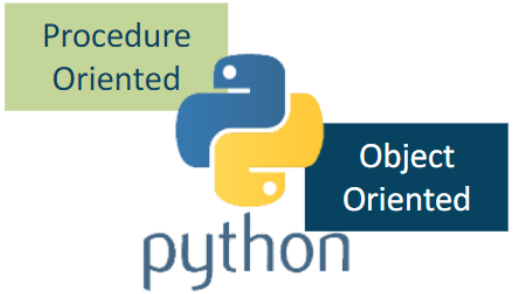
Portable

Supported by many platforms like Linux, Windows, FreeBSD, Macintosh, Solaris, BeOS, OS/390, PlayStation, Windows CE, etc.

Why Python?

Supports different Programming Paradigm

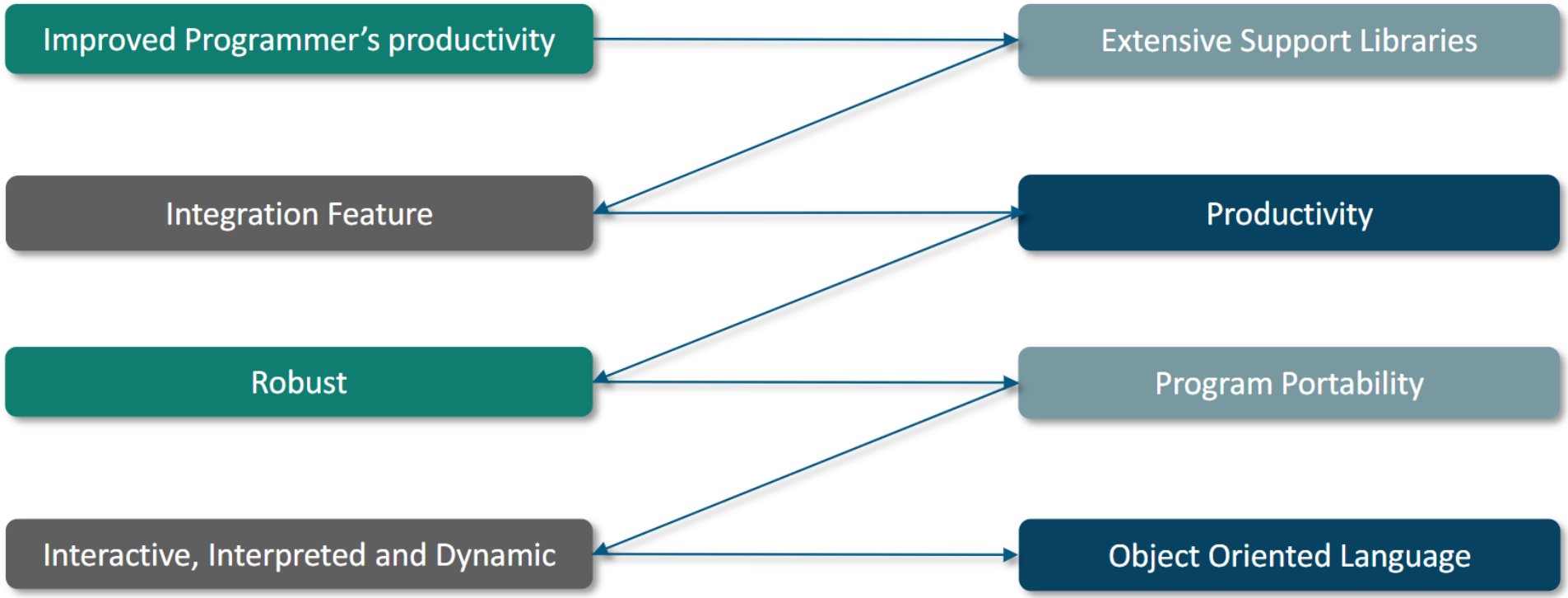
Python supports procedure-oriented programming as well as object-oriented programming



Extensible

Python code can invoke C and C++ libraries, can be called from and C++ programs, can integrate with Java and .NET components

Why Python?



Companies Using Python

edureka!

Who is using Python?

The popular YouTube video sharing system is largely written in Python

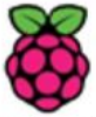
Google makes extensive use of Python in it's web search system

Dropbox storage service codes both its server and client software primarily in Python

The Raspberry Pi single-board computer promotes Python as its educational language

Google

Dropbox



RaspberryPi

COMPANIES USING PYTHON

BitTorrent

NETFLIX

BitTorrent peer-to-peer file sharing system began its life as a Python Program

NASA uses Python for specific Programming Task

The NSA uses Python for cryptography and intelligence analysis

Netflix and Yelp have both documented the role of Python in their software infrastructures

Applications Where Python Is Used

edureka!

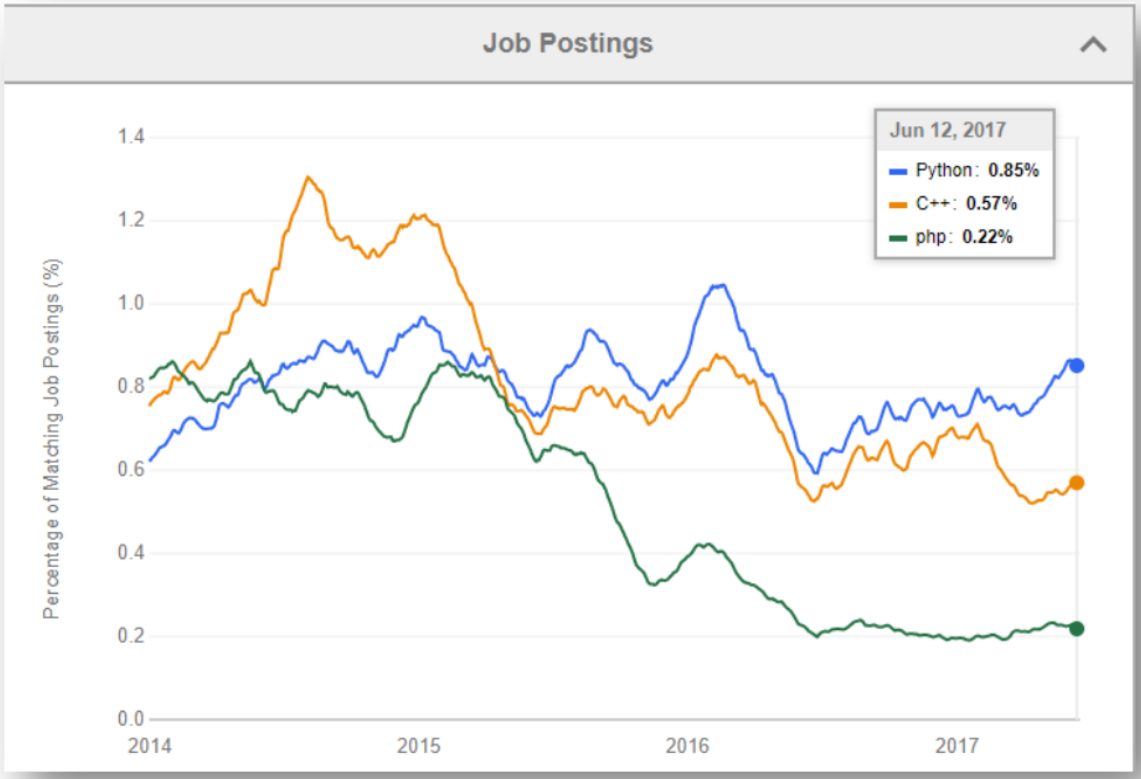
Python Applications

Are there enough jobs in the market for Python related roles

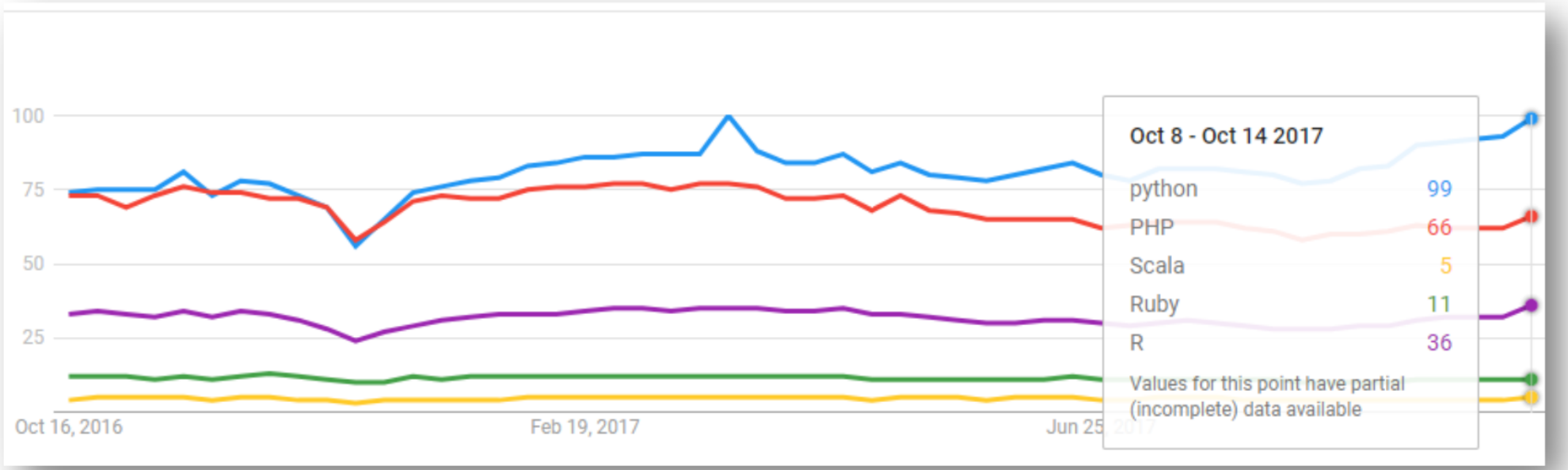
To prove that, I will show you few stats

Job Trends

Job trends from Indeed.com shows remarkable increase in the demand of Python related jobs



Python Graph on Google Trends



Python Interpreter

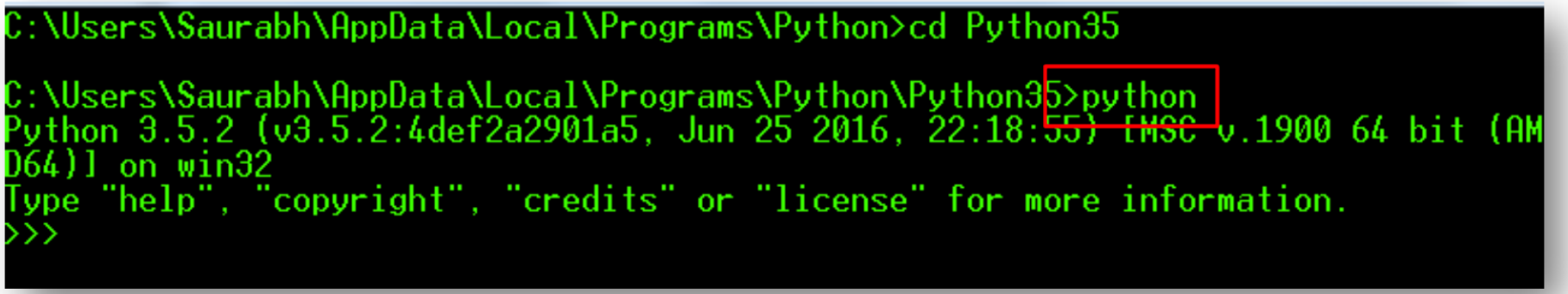
Python Interpreter

- Python Interpreter is a program that reads and executes code
- This includes source code, pre-compiled code and scripts
- Example: **help('for')**

```
>>> help('for')
The "for" statement
*******************

The "for" statement is used to iterate over the elements of a sequence
(such as a string, tuple or list) or other iterable object:

   for_stmt ::= "for" target_list "in" expression_list ":" suite
                ["else" ":" suite]

The expression list is evaluated once; it should yield an iterable
object.  An iterator is created for the result of the
"expression_list".  The suite is then executed once for each item
provided by the iterator, in the order of ascending indices.  Each
item in turn is assigned to the target list using the standard rules
for assignments, and then the suite is executed.  When the items are
exhausted (which is immediately when the sequence is empty), the suite
in the "else" clause, if present, is executed, and the loop
terminates.
```

Getting Started With Python in Command Prompt

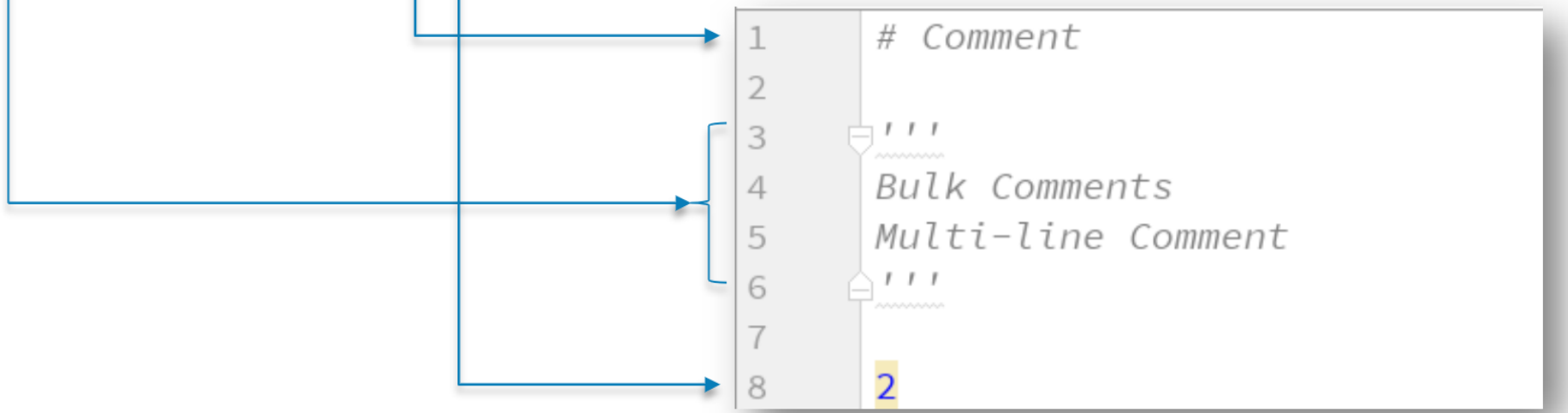
You need to write python in command prompt for automatic installation of all packages. Once this is done, you can write your code



Comments and Literals

Comments: Any text to the right of the # symbol is mainly used as notes for the readers. Statements on right side of # does not get executed. It gives more information about function

- **Bulk Comments:** Enclose the code in triple quoted strings (“””)
- **Literal Constants:** Any number or character, or set of characters



Indentation

No braces to indicate blocks of code for class and function definitions or flow control

Blocks of code are denoted by line indentation, which is rigidly enforced

The number of spaces in the indentation is variable, but all statements within the block must be indented the same amount

Leading whitespace at the beginning of a logical line is used to compute the indentation level of the line, which in turn, is used to determine the grouping of statements

Python Code Execution

Source

The structured code is written and saved with .py extension

Byte Code

The code is converted into Byte code for machine to understand

PVM

The Byte codes are executed

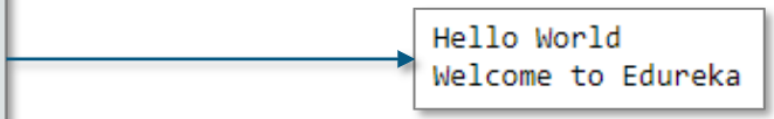
Demo: Creating “Hello World” Program

Output

Every character in Python should be enclosed within single or double quotes

Output after running new.py

```
print('Hello World')
print("Welcome to Edureka")
```



```
Hello World
Welcome to Edureka
```

Variables

Identifier

- A Python Identifier is a name used to identify a variable, function, class, module or other object
- An identifier starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores and digits (0 to 9)
- Python is a case sensitive programming language
- Python does not allow special characters such as @, $ and % within identifiers

Identifiers – Naming Conventions

Class names start with an uppercase letter. All other identifiers start with a lowercase letter

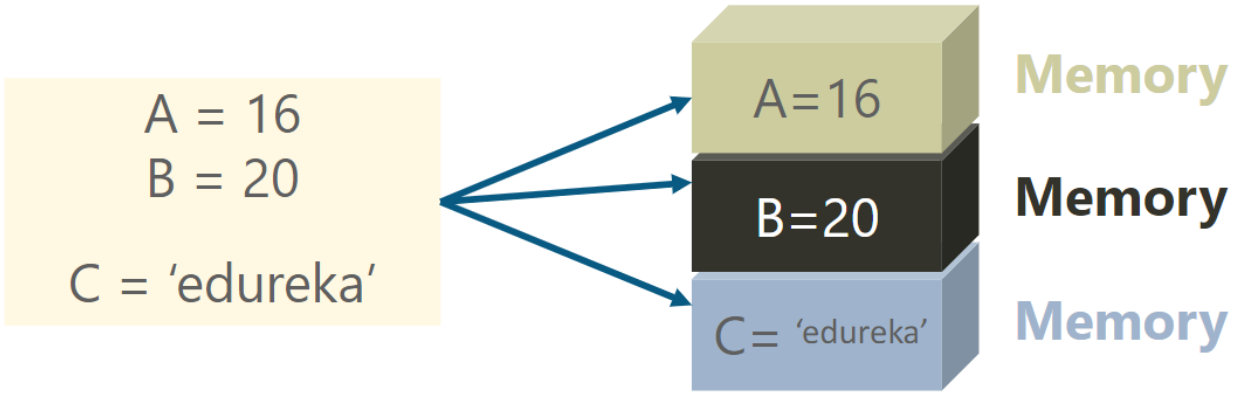
Starting an identifier with a single leading underscore indicates that the identifier is private

Starting an identifier with two leading underscores indicates a strongly private identifier

If the identifier also ends with two trailing underscores, the identifier is a language-defined special name

Variables

Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory



Variables (Contd.)

Consider the below screenshot, it explains how to assign a value to a *Variable*

Assigning values 10 and edureka! to variables A and B respectively

Right click and click on 'Run File_name' to execute the code

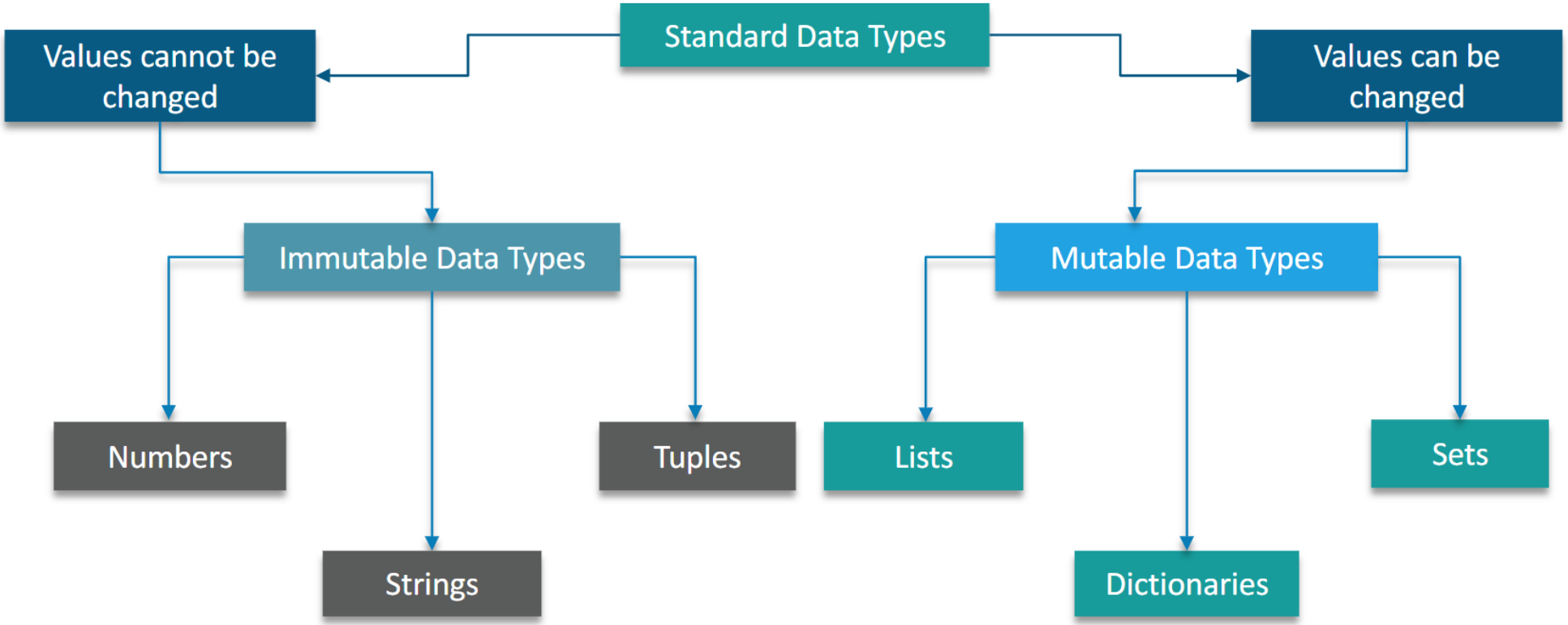
```
A=10
B='edureka!'
print(A,B)
```

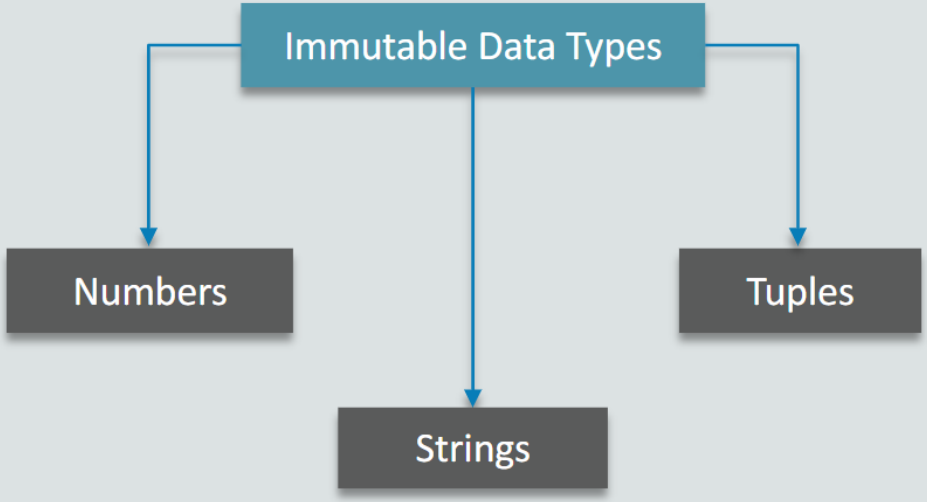
Output

```
10 edureka!
```

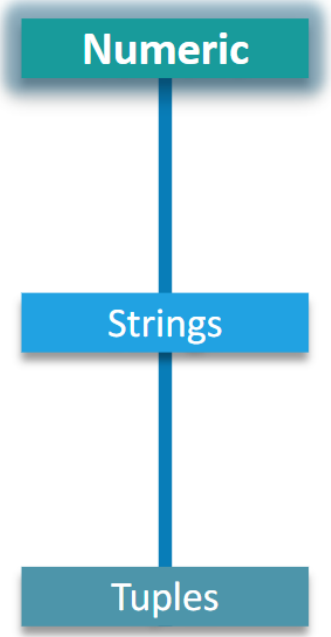
Standard Data Types

Standard Data Types





Numeric Data Type – Immutable Data Type



Python supports three different numerical values

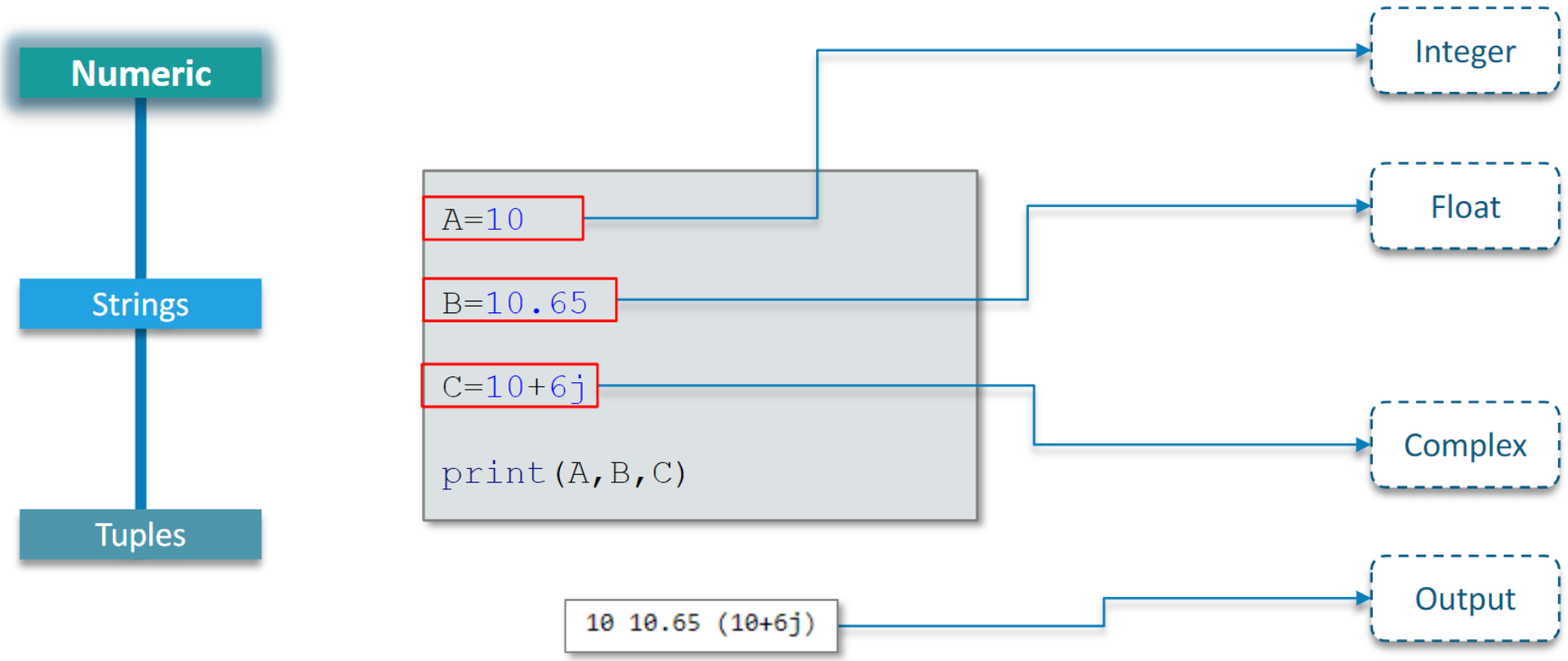
- int (Signed Integers)
- Complex Numbers
- float (Real Numbers)

In Python you can represent **Numbers** in multiple ways:

- *Binary*
- *Octal*
- *Hexadecimal*

edureka!

Numeric Data Type – Immutable Data Type



Strings – Immutable Data Type

Numeric

Strings

Tuples

The continuous set of characters represented within quotation is called as String. Python allows for either pairs of single or double quotes. Python does not support a character type, these are treated as strings of length one

```
A='Welcome To edureka!'

B="Python is Great"

print(A)

print(B)
```

Python cannot differentiate between single and double quotes

```
Welcome To edureka!
Python is Great
```

Output

Tuples – Immutable Data Type

Tuples consists of a number of values separated by comma. It is enclosed within parenthesis

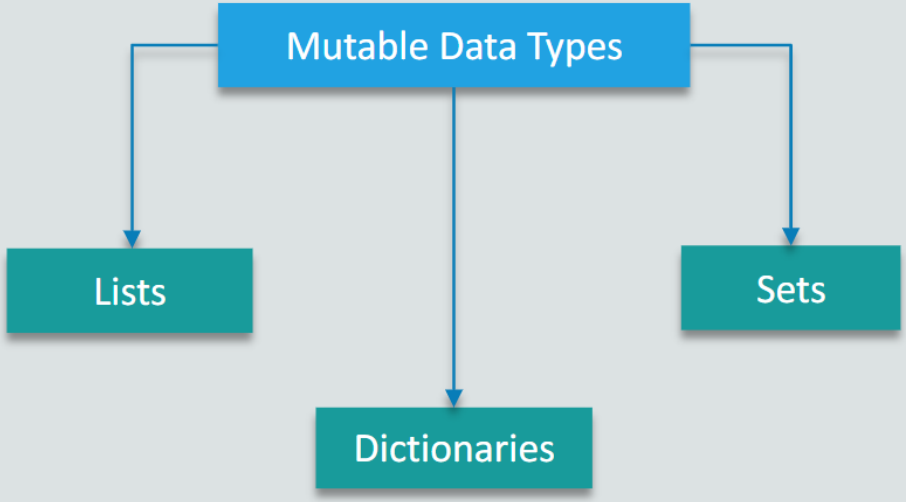
- Numeric
- Strings
- **Tuples**

```
A=(1,2,3.15,'edureka!')

print(A)
```

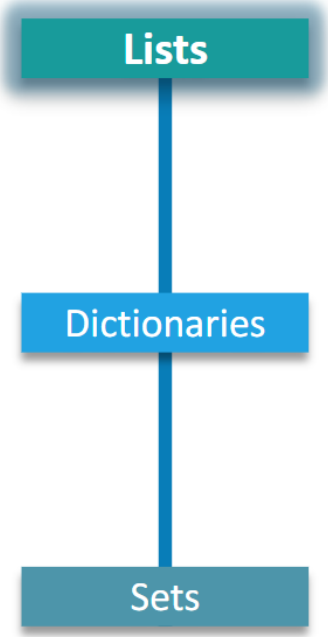
A Tuple can have objects of different data types, unlike Arrays in 'C'

```
(1, 2, 3.15, 'edureka!')
```

Output



Lists - Mutable Data Type



List is an ordered set of elements enclosed within square brackets. The main differences between Lists and Tuples are:

- Lists are enclosed in brackets[] and Tuples are enclosed within parenthesis()
- Lists are Mutable and Tuples are Immutable
- Tuples are faster than Lists

```
A=[1,2,3.15,'edureka!']

print(A)
```

Lists are enclosed within square brackets

```
[1, 2, 3.15, 'edureka!']
```

Output

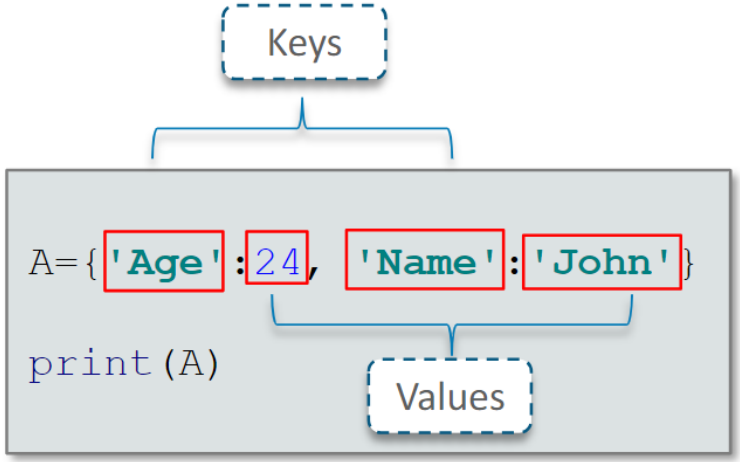
Dictionaries - Mutable Data Type

Lists

Dictionaries

Sets

Dictionaries contain key value pairs. Each key is separated from its value by a colon (:), the items are separated by comma, and the whole thing is enclosed within curly braces




```
{'Age': 24, 'Name': 'John'}
```

Output

Sets - Mutable Data Type

Lists

Dictionaries

Sets

A set is an unordered collection of items. Every element is unique. A set is created by placing all the items (elements) inside curly braces {}, separated by comma.

Every element in a Set has to be unique

```
A={1,2,3,3}

print(A)
```

```
{1, 2, 3}
```

Output – Notice that 3 has appeared only once

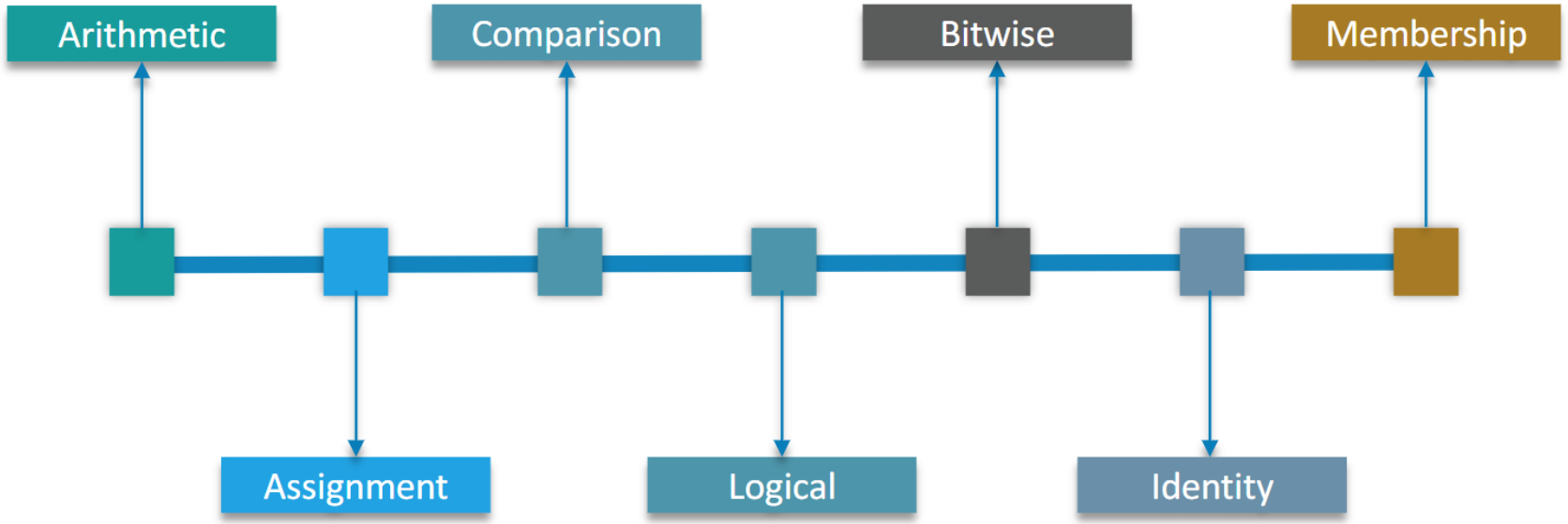
You can also create a Set by calling an in built-function '**set**'

edureka!

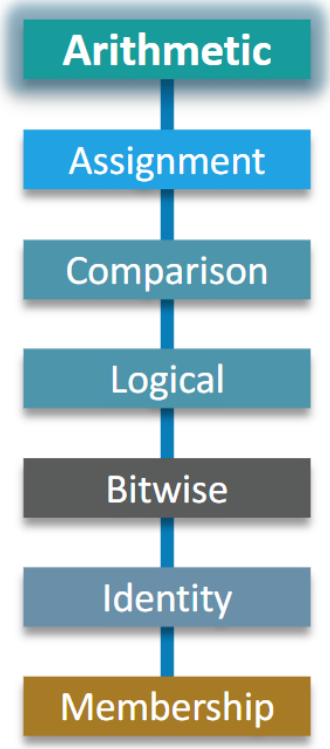
Python Operators

Operators

Operators are the constructs which can manipulate the values of the Operands. Consider the expression 2 + 3 = 5, here 2 and 3 are Operands and + is called Operator

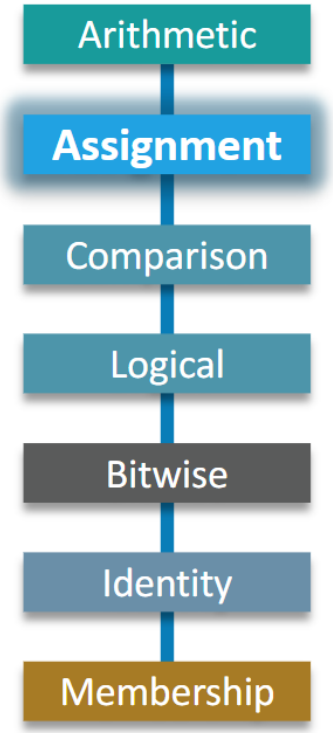


Arithmetic Operators



Addition	a + b
Subtraction	a – b
Multiplication	a * b
Division	a / b
Modulus	a % b
Exponent	a ** b
Floor Division	a // b

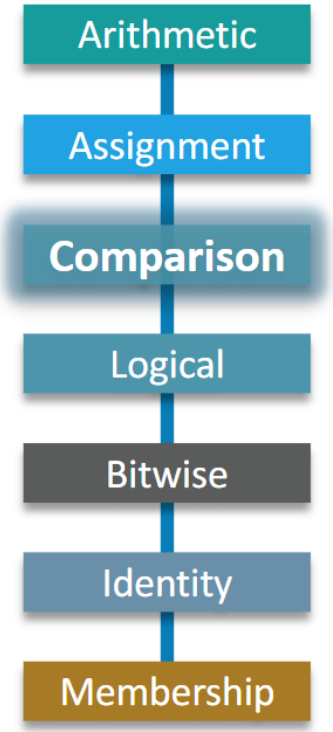
Assignment Operators



Assigns value from right to left	a = b
a = a + b	a + = b
a = a - b	a - = b
a = a*b	a * = b
a = a/b	a /= b
a = a**b	a** = b
a=a//b	a//=b

edureka!

Comparison Operators



Equal To	a == b
Not Equal To	a != b
Greater Than	a > b
Less Than	a < b
Greater Than Equal To	a >= b
Less Than Equal To	a <= b

edureka!

Logical Operators

- Arithmetic
- Assignment
- Comparison
- **Logical**
- Bitwise
- Identity
- Membership

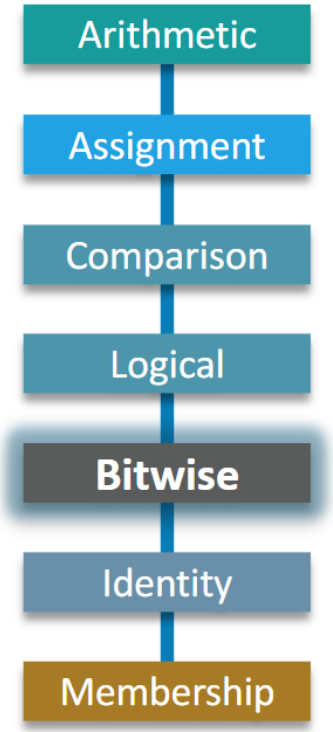
a and b → Returns a, if a is False, b otherwise

a or b → Returns b, if b is False, a otherwise

not a → Returns True, if a is False, False otherwise

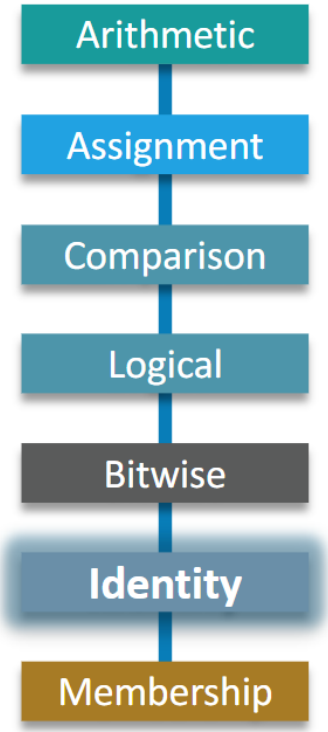
edureka!

Bitwise Operators



Binary AND	a & b
Binary OR	a | b
Binary XOR	a ^ b
Binary NOT	a ~ b
Binary Left Shift	a <<
Binary Right Shift	a >> b

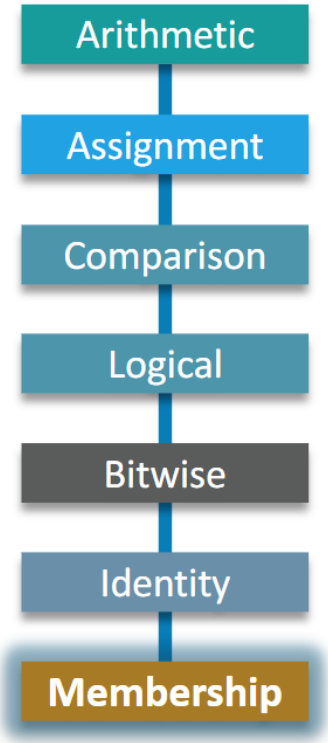
Identity Operators



is → Evaluates to TRUE, if the variables on either side of the operator point to the same object and FALSE otherwise

is not → Evaluates to FALSE, if the variables on either side of the operator point to the same object and TRUE otherwise

Membership Operators

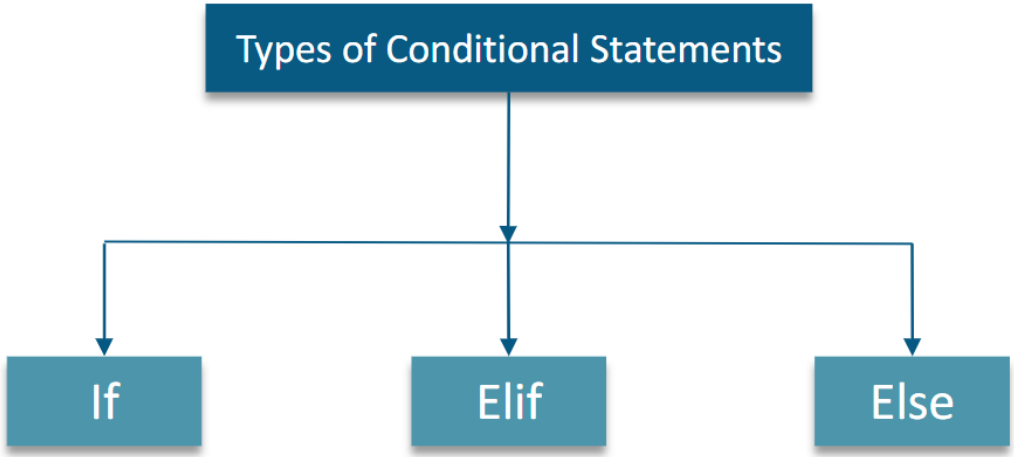


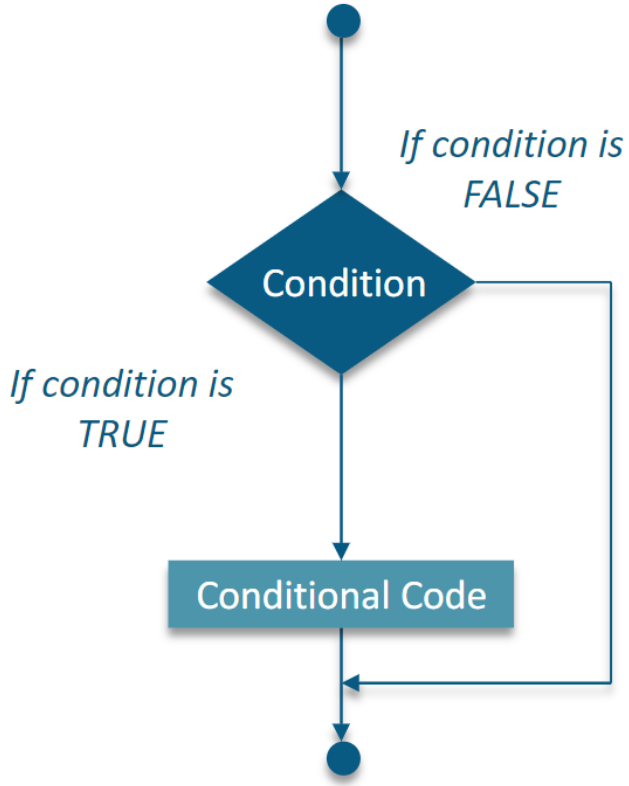
in	Evaluates to TRUE, if it finds a variable in the specified sequence and FALSE otherwise
not in	Evaluates to TRUE, if it does not find a variable in the specified sequence and FALSE otherwise

Conditional Statements

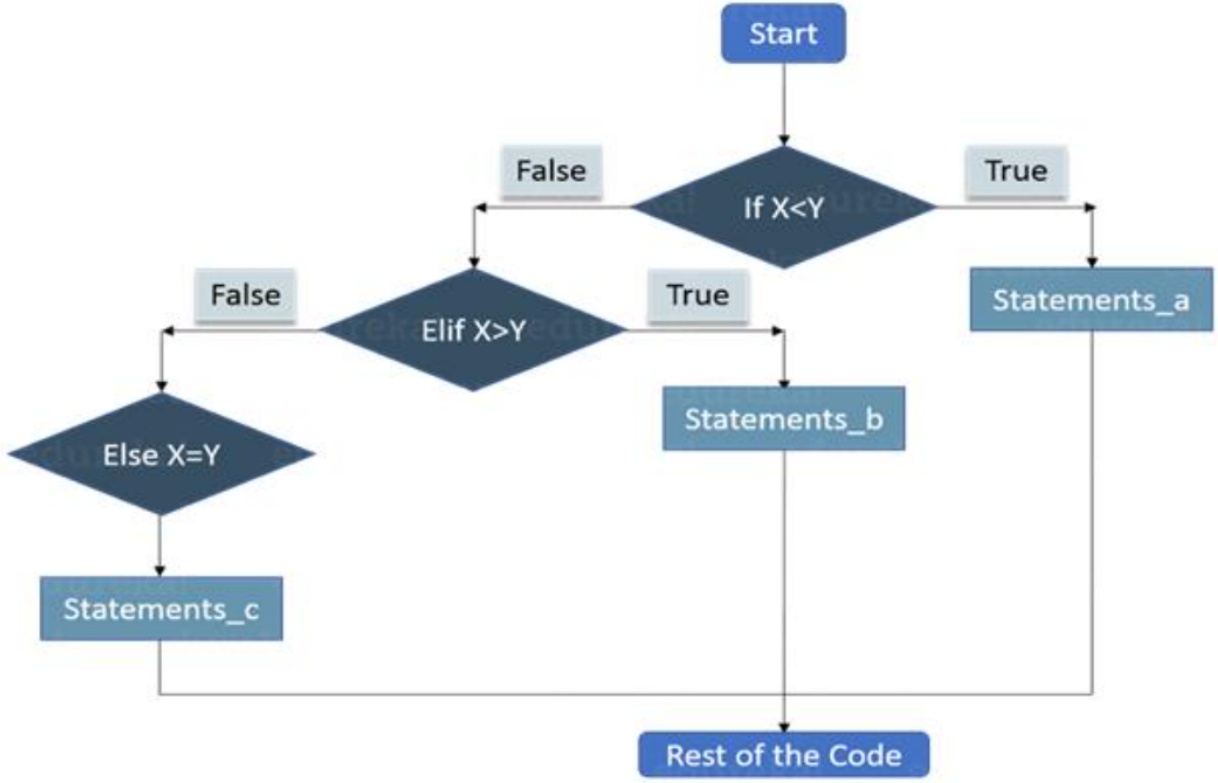
Conditional Statements

Conditional statements are used to execute a statement or a group of statements, when some condition is true





If Elif Else Statements



If, Elif and Else Statements

Consider the syntax and example below:

Syntax:

```
if condition1:
    statements

elif condition2:
    statements

else:
    statements
```

Example:

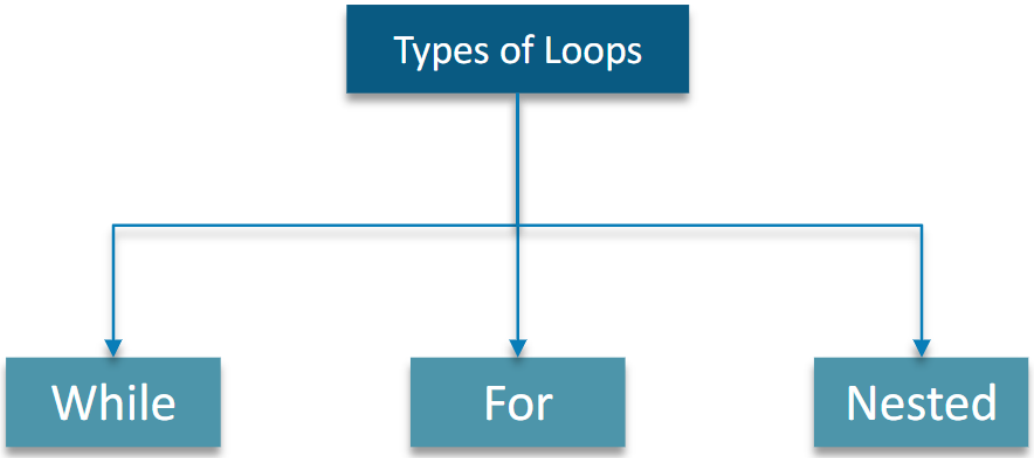
```
X=10
Y=12

if(X<Y):
    print('X is less than Y')
elif(X>Y):
    print('X is greater than Y')
else:
    print('X and Y are equal')
```

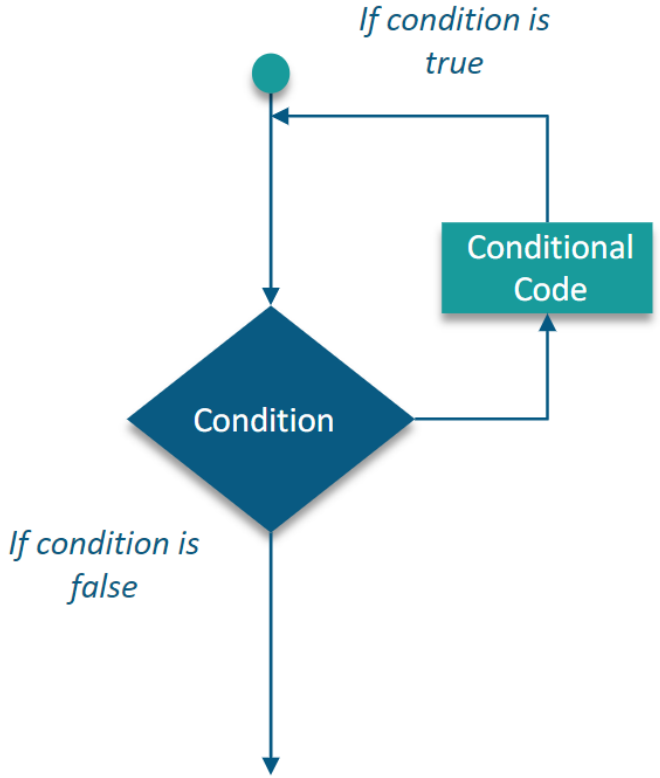
```
X is less than Y
```

Loops

Loops

A loop statement allows us to execute a statement or a group of statement multiple times



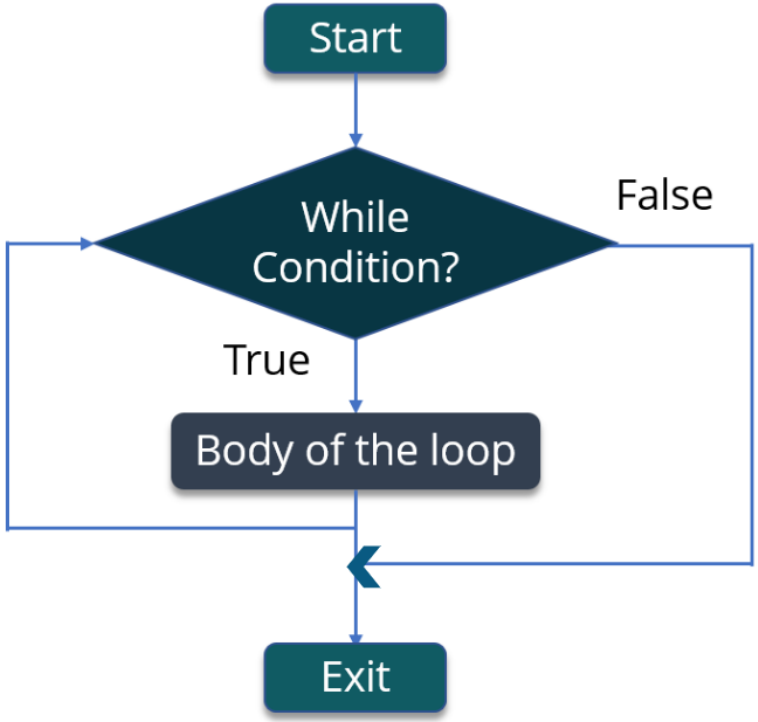


While Loop

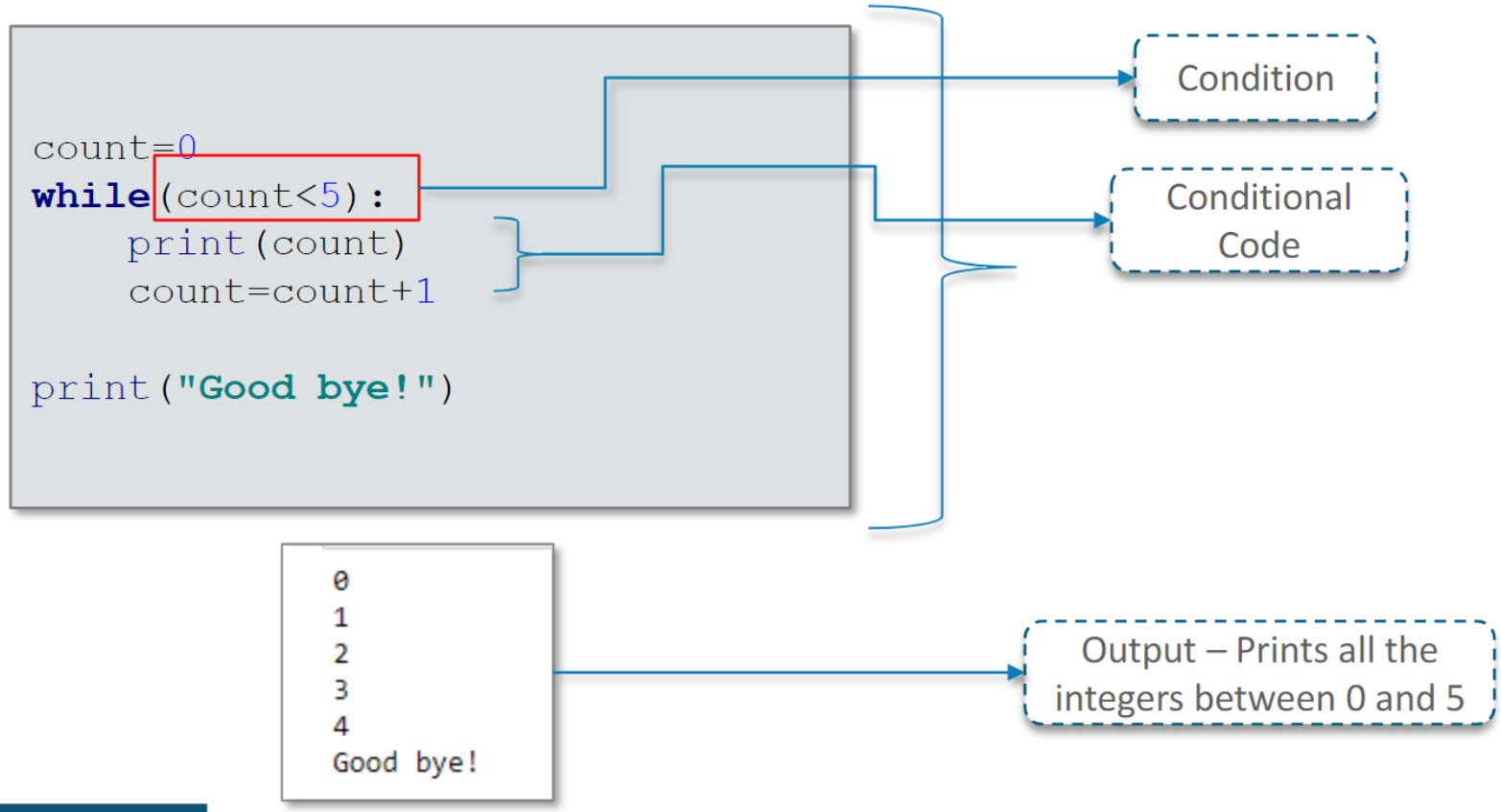
"*While*" loops are known as indefinite or conditional loops. They will keep iterating until certain conditions are met. There is no guarantee ahead of time regarding how many times the loop will iterate

Syntax:

```
1  while expression:
2      statements
```



While Loop Example



While Loop Example

A little guessing game

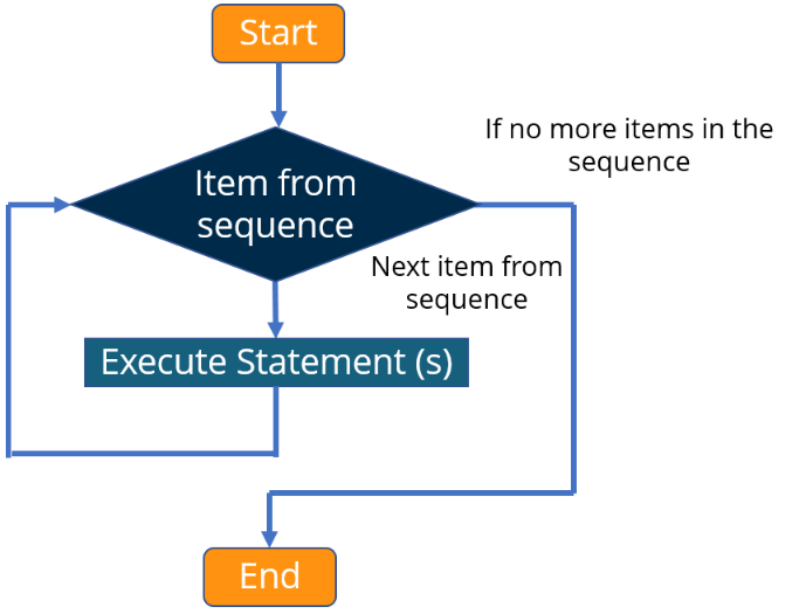
Let the correct answer be 13

Input Number: 10	Number is too small
Input Number: 15	Number is too large
Input Number: 13	Exit: Congratulations. You made it!

For Loop

- "*For*" loop is a Python loop which repeats a group of statements a specified number of times. The for loop provides a syntax where the following information is provided:
  - Boolean condition
  - The initial value of the counting variable
  - Incrementation of counting variable

```
1  for <variable> in <range>:
2    stmt1
3    stmt2
4    ...
5    stmtn
```



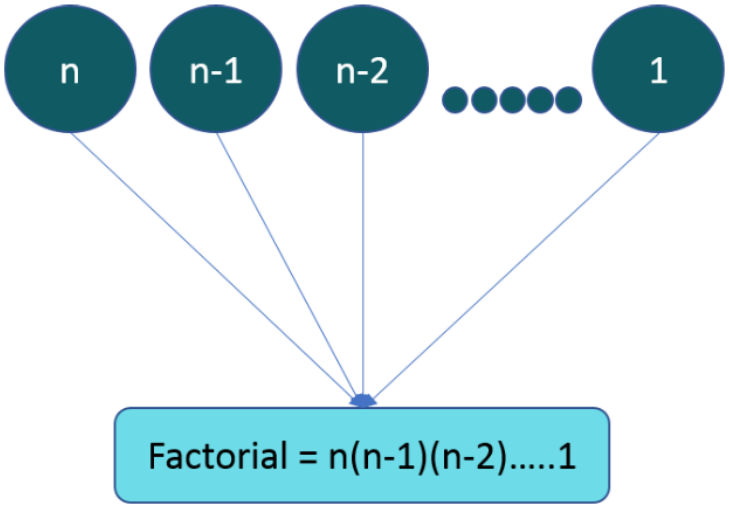
For Loop Example

Difference between For and While loop is that, in While loop we don't know the amount of iterations, where as in For loop we are aware of how many times the block of code will be executed

```
fruits=['Banana','Apple','Grapes']

for index in range(len(fruits)):
    print(fruits[index])
```

For Loop Example

We will be using For loop to write a program that calculates the factorial of any number



Example

3! = 3(2)(1)

4! = 4(3)(2)(1)

5! = 5(4)(3)(2)(1)

Nested Loops

Nested Loop, basically means a loop inside a loop. It can be a For loop inside a While loop and vice-versa. It can also be a While loop inside a While loop or For loop inside a For loop

```
count=1
for i in range(10):
    print(str(i)*i)

    for j in range(0,i):
        count=count+1
```

For loop inside a *For* loop

```
1
22
333
4444
55555
666666
7777777
88888888
999999999
```

Output

Nested Loops Example

Lets code a program in Python that effectively simulates a bank ATM.

Enter the 4-digit pin

- Check balance
- Make a withdrawal
- Pay in
- Return card

Loop Control Statements

- Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed

Control Statement	Description
break statement	Terminates the loop statement and transfers execution to the statement immediately following the loop
continue statement	Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating
pass statement	The pass statement in Python is used when a statement is required syntactically but you do not want any command or code to execute

Example - Loop Control Statements

```
for i in range(10,50):
    print(i)
    if(i==30):
        break
```

```
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
```

```
for j in range(1,11):
    print(j)
    if(j==5):
        continue
```

```
1
2
3
4
5
6
7
8
9
10
```

```
for k in range(1,3):
    pass
print("Loop ends here")
```

```
Loop ends here
```

Command Line Parameters

Command Line Parameters

- It is possible to pass **arguments** to Python programs when they are executed
- The brackets which follow main are used for this purpose
- . argv refers to the number of **arguments** passed, and argv[] is a pointer array which points to each **argument** which is passed to main
- The Python **sys** module provides access to any command-line arguments via the **sys.argv**. This serves two purposes :
  - sys.argv is the list of command-line arguments
  - len(sys.argv) is the number of command-line arguments

Command Line Parameters(Contd.)

- **Example:**
  - Consider the following script test.py

```
#!/usr/bin/python
import sys
print ('Number of arguments:',
len(sys.argv), 'arguments.' )
print ('Argument List:', str(sys.argv) )
```

  - Now, run above script as follow:

```
$ python test.py arg1 arg2 arg3
```

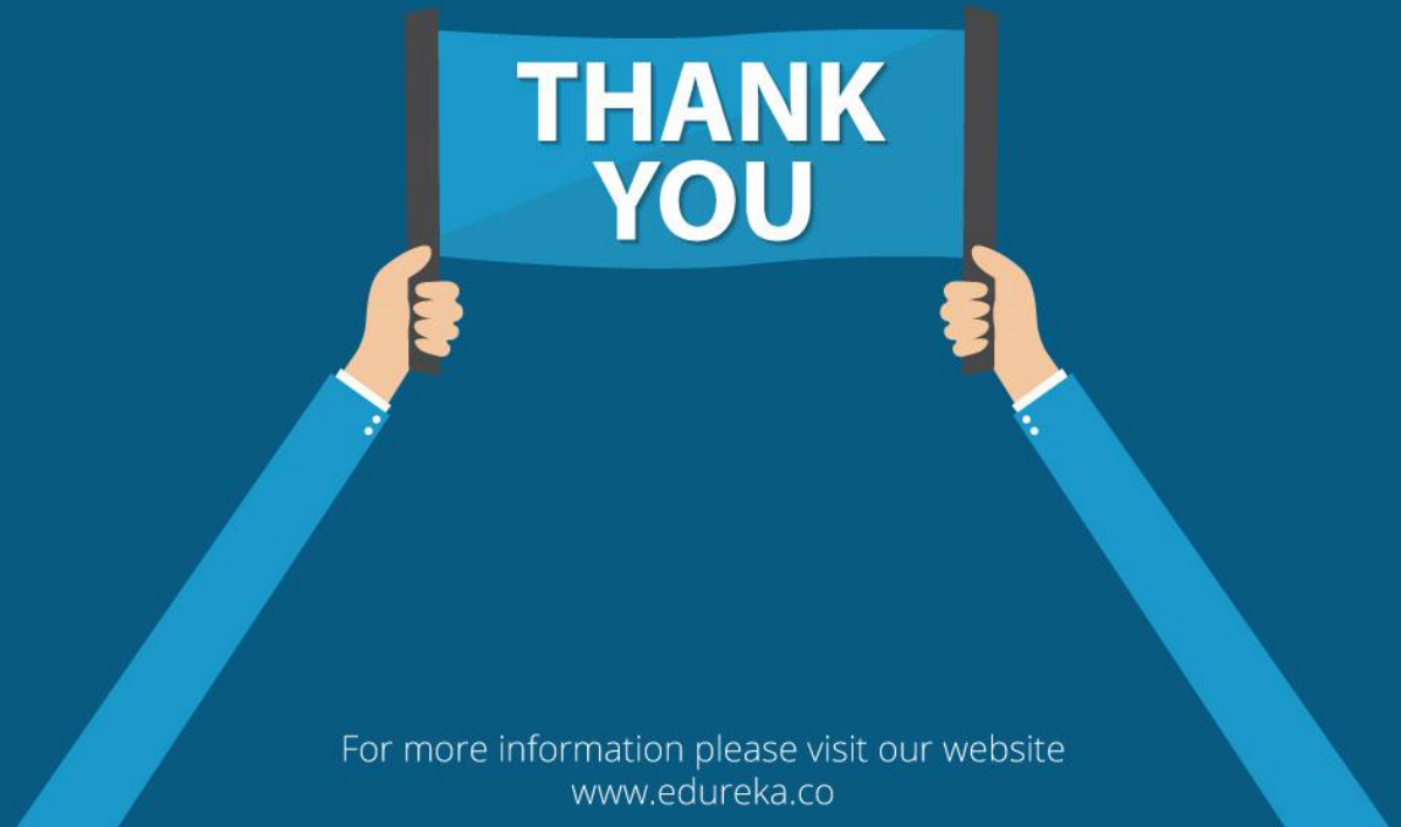
  - After running this script, Output will be

```
Number of arguments: 4 arguments. Argument
List: ['test.py', 'arg1', 'arg2', 'arg3']
```

Summary

In this module, you should have learnt:

- Why we are using Python over other languages
- Python IDEs
- Identifiers, Variables
- Standard Data types
- Operators
- Conditional Statements
- Loops and Iterations
- Command Line Parameters



For more information please visit our website
www.edureka.co