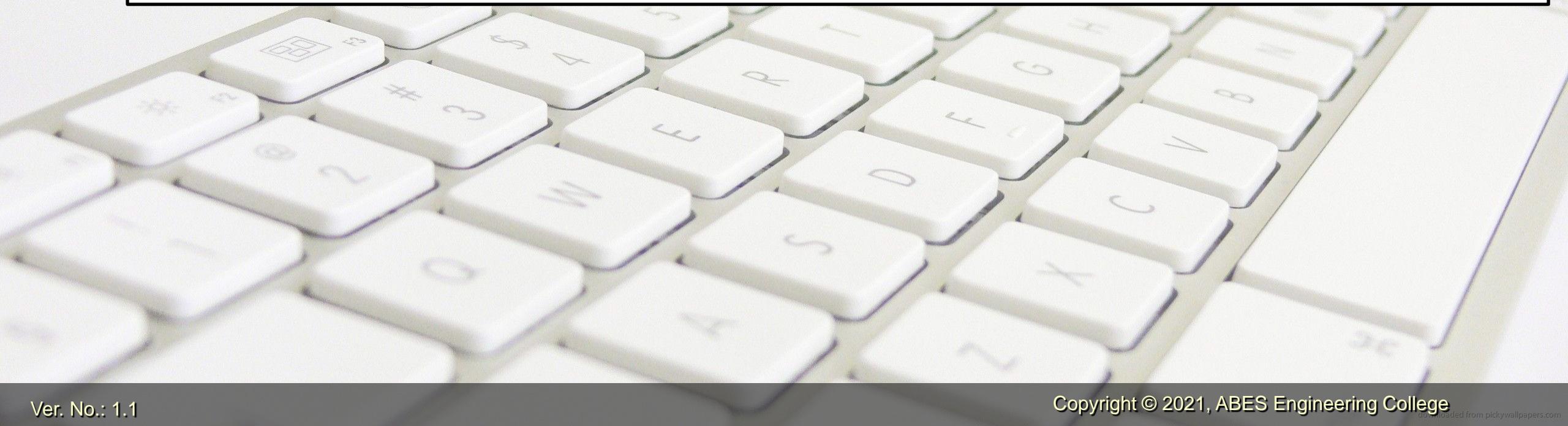


# 1.1 Getting started with Python

**Day 1**



# General Guideline

© (2021) ABES Engineering College.

This document contains valuable confidential and proprietary information of ABESEC. Such confidential and proprietary information includes, amongst others, proprietary intellectual property which can be legally protected and commercialized. Such information is furnished herein for training purposes only. Except with the express prior written permission of ABESEC, this document and the information contained herein may not be published, disclosed, or used for any other purpose.

# Topics Covered

Day 1	Day 2	Day 3	Day 4
<p>1.1 Getting started with Python</p> <ul style="list-style-type: none"><li>• 1.1.1 Introduction to programming and coding</li><li>• 1.1.2 Why choose Python</li><li>• 1.1.3 Scope of Python</li><li>• 1.1.4 Python History</li><li>• 1.1.5 Python Features</li></ul>	<p>1.1 Getting started with Python</p> <ul style="list-style-type: none"><li>• 1.1.6 Advantages of Python</li><li>• 1.1.7 Disadvantages of Python</li><li>• 1.1.8 Applications of Python</li><li>• 1.1.9 Different Flavors of Python</li><li>• 1.1.10 Different Python Frameworks</li><li>• 1.1.11 Python in contrast with other programming languages</li></ul>	<p>1.2 Python Installation Guide</p> <ul style="list-style-type: none"><li>• 1.2.1 Introduction to python IDE – IDLE</li><li>• 1.2.2 Setting Up Your Environment</li><li>• 1.2.3 Installation of Python and Anaconda Navigator</li><li>• 1.2.4 Quick Tour of Jupyter Notebook</li><li>• 1.2.5 Python vs. IPython</li><li>• 1.2.6 Online compilation support</li><li>• 1.2.7 Running python script using command prompt</li></ul>	<p>1.3 Basics of Python</p> <ul style="list-style-type: none"><li>• 1.3.1 Python keywords</li><li>• 1.3.2 Python Statement and Comments</li><li>• 1.3.3 Python Literals</li><li>• 1.3.4 Data Types</li><li>• 1.3.5 Variables</li><li>• 1.3.6 type (), dir (),ID command</li></ul>

# Topics Covered

Day 5

## 1.3 Basics of Python

- 1.3.7 Type conversion: implicit and explicit
- 1.3.8 Basic I/O Operations: `input()`, `print()`

Day 6

## 1.3 Basics of Python

- 1.3.9 Operators
- 1.3.10 Precedence and associativity
- 1.3.11 Python 2 vs Python 3

# Session Plan - Day 1

## 1.1 Getting started with Python

1.1.1 Introduction to programming and coding

1.1.2 Why choose Python

1.1.3 Scope of Python

1.1.4 Python History

1.1.5 Python Features

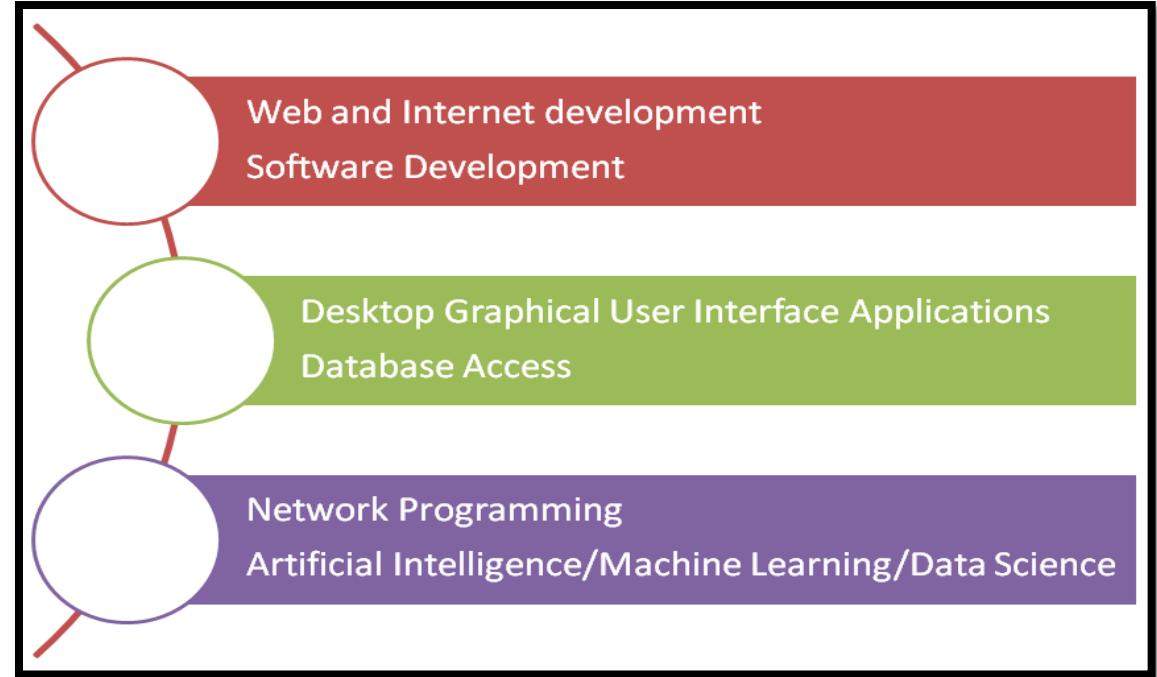
# Python is a Popular Programming language

?

<https://www.wordclouds.com/>

# Some Areas where it is Popular

- Application Development
- Web Development
- Artificial Intelligence
- Data Science



# Why choose Python : Lets Discuss More

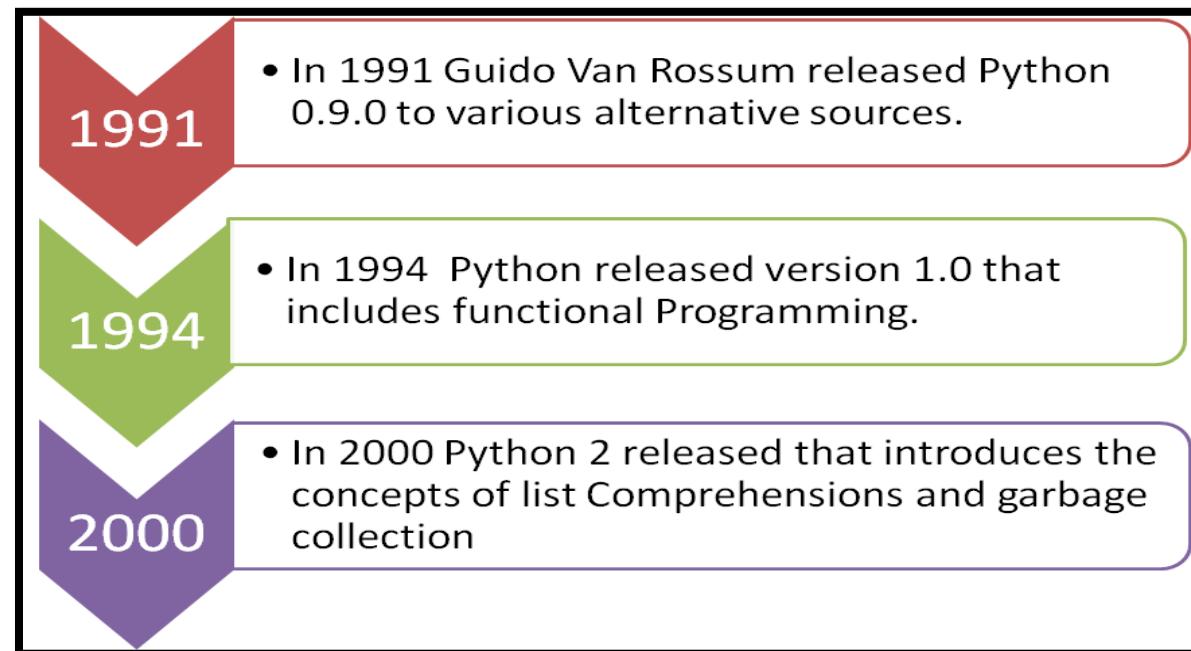
- Simple and Easier to learn
- Good Readability
- Free and Open Source
- Python is a platform-independent language
- High Level and Interpreted language
- Extensive libraries: Python has a vast number of libraries.

# Scope of Python

- ❑ Python Language provides promising and rewarding career in the IT industry
- ❑ Various Job roles advanced in Python with high paying jobs:
  - ❑ Research Analyst
  - ❑ DevOps Engineer
  - ❑ Python Developer
  - ❑ Data Analyst
  - ❑ Software Developer
  - ❑ Game Developer
  - ❑ Web Scrapper

# Python History

Python was written in the late 1980s by Guido van Rossum at Centrum Wiskunde & Informatica (CWI)



# Python Features

Easy to learn  
and use

GUI  
Programming  
Support

Large  
Standard  
Library

Dynamically  
typed

Interpreted  
Language

Free and  
Open Source

Object  
Oriented

Cross  
Platform  
Language

Expressive  
Language

# Review Questions

➤ In which year was the Python language developed?

- 1995
- 1972
- 1981
- 1989

➤ Who developed the Python language?

- Zim Den
- Guido van Rossum
- Niene Stom
- Wick van Rossum

# Review Questions

- How many keywords are there in python 3.7?
  - 32
  - 33
  - 35
  - 30
- Which one of the following is the correct extension of the Python file?
  - .py
  - .python
  - .p
  - None of these

# Session Plan - Day 2

## 1.1 Getting started with Python

**1.1.6 Advantages of Python**

**1.1.7 Disadvantages of Python**

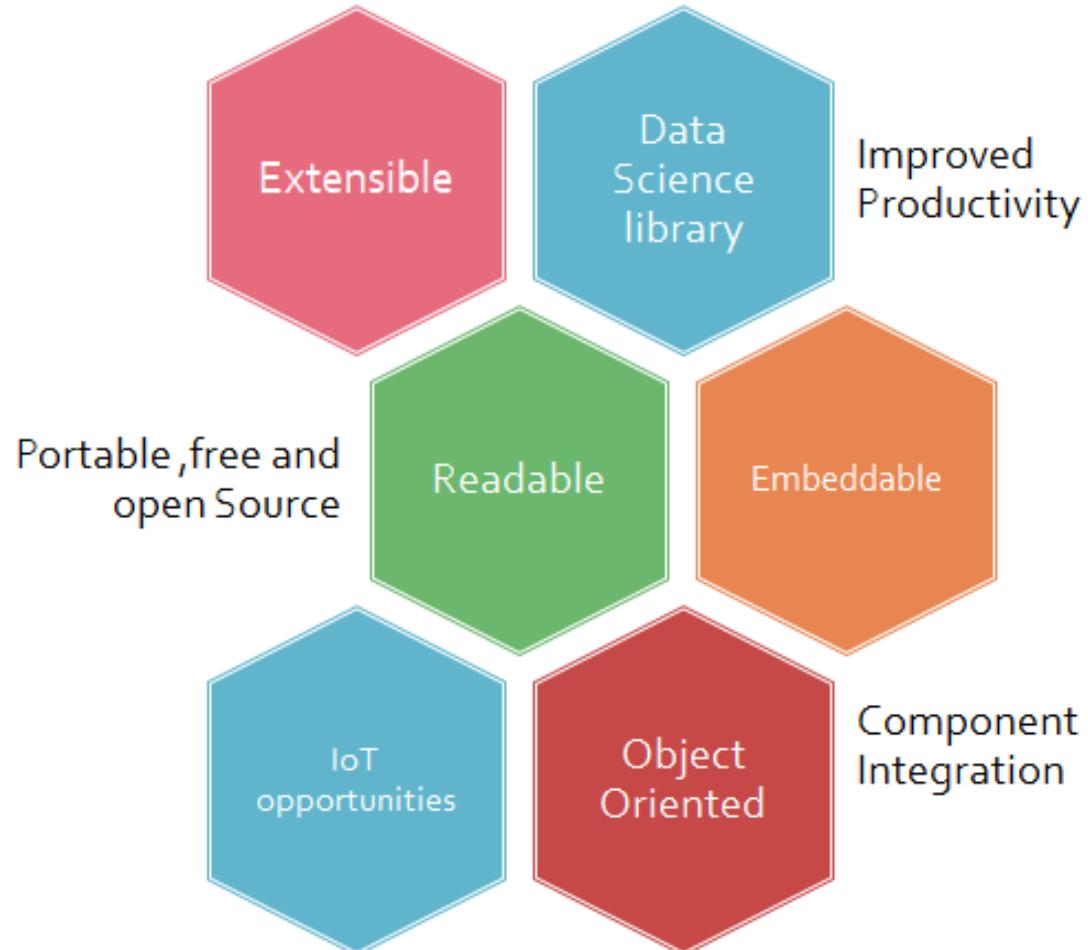
**1.1.8 Applications of Python**

**1.1.9 Different Flavors of Python**

**1.1.10 Different Python Frameworks**

**1.1.11 Python in contrast with other programming languages**

# Advantages of Python



# Contd..

Write a Program using any language to print “ HELLO WORLD “.

## Java Program :

```
public class Hello
{
    public static void main(String argv[])
    {
        System.out.println("Hello, World!");
    }
}
```

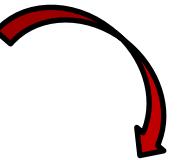
## C++ Program :

```
#include <iostream><br>
int main()
{
    cout << "Hello World" << endl;
    return 0;
}
```

# Contd..

In Python

**print ( "Hello World")**



# Disadvantages of Python

- ❑ Python code is executed line by line since Python is interpreted as **slower in runtime** than other programming languages like C++, Java, and PHP.
- ❑ Python takes a **lot of memory** due to the flexibility of the data types, so it is not a desirable choice for memory-intensive tasks.
- ❑ Although Python serves as an excellent server-side programming language, it is **less commonly used to build intelligent phone-based Applications**.
- ❑ Python is **rarely used in Enterprise development** because Python has some limitations with Database Access compared to primarily other used technologies like JDBC (Java Database Connectivity) and ODBC (Open Database Connectivity) as Python Language Database layers are underdeveloped.

# Your Future....!!!

## Top Companies Using Python



JPMorganChase



YouTube

pandora

Bitbucket



shutterstock



Prezi

Vine



Udemy



BankofAmerica.



Google



Quora



IBM

@mailgun

moz://a



amazon

Expedia



DISQUS



MIT

Massachusetts  
Institute of  
Technology

reddit

hike

YAHOO!



trivago

NOKIA

SendGrid

redis

theONION



# Applications of Python

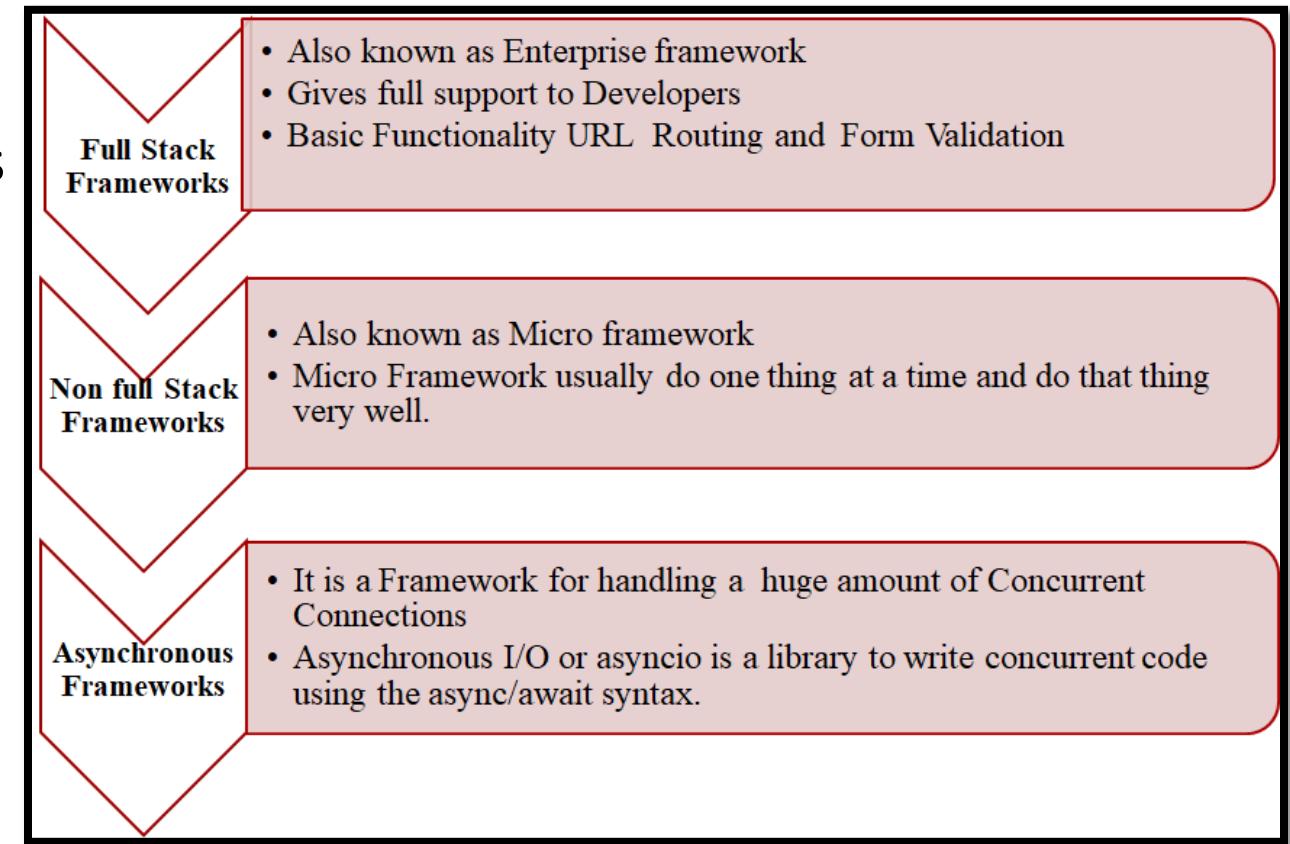
- ❑ Web and Internet Development
- ❑ Game Development
- ❑ Desktop GUI Applications
- ❑ Artificial Intelligence and Machine Learning
- ❑ Data Science and Data Visualization
- ❑ Web Scraping Applications
- ❑ Desktop Applications
- ❑ Business Applications
- ❑ Image Processing and Computer Graphics
- ❑ Language Development
- ❑ Popular Applications Built on Python

# Different Flavors of Python

- Cpython
- Jpython
- Active Python
- Anaconda Python
- PyPy
- Win Python
- Python Portable

# Different Python Frameworks

- Full-Stack Frameworks
- Non-Full Stack Frameworks
- Asynchronous Frameworks



# Review Questions

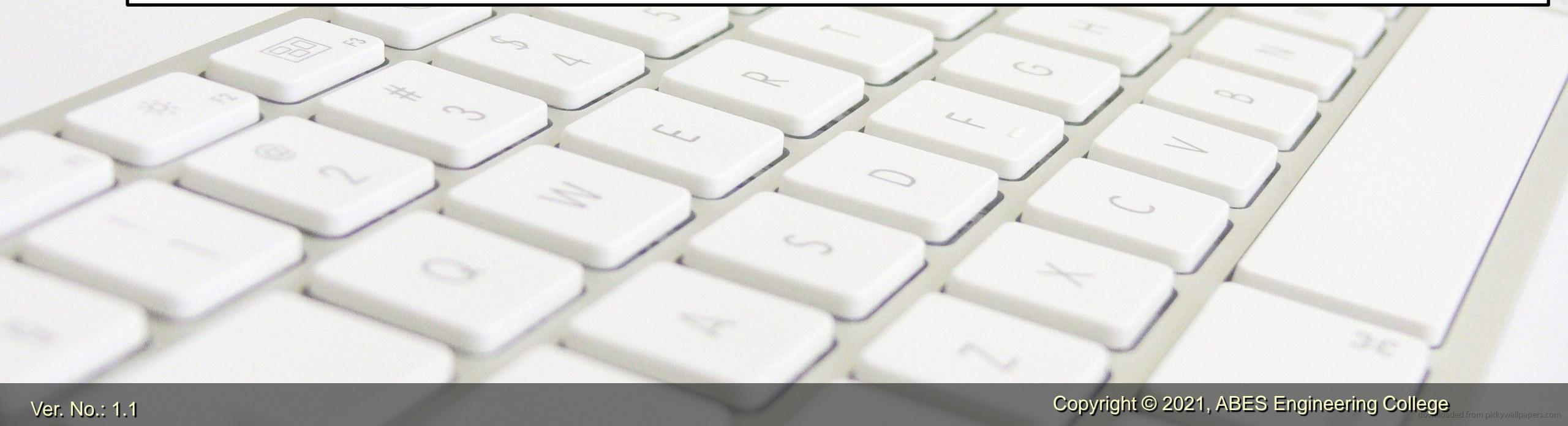
- How to output the string “May the odds favor you” in Python?
  - `print("May the odds favor you")`
  - `echo("May the odds favor you")`
  - `System.out("May the odds favor you")`
  - `printf("May the odds favor you")`
  
- In which year was the Python 3.0 version developed?
  - 2005
  - 2000
  - 2010
  - 2008

# Review Questions

- Python is often described as a:
  - Batteries excluded language
  - Gear included language
  - Batteries included language
  - Gear excluded language
  
- What do we use to define a block of code in Python language?
  - Indentation
  - Key
  - Brackets
  - None of these

# 1.2 Python Installation Guide

**Day 3**



# Session Plan - Day 3

## 1.2 Python Installation Guide

- **1.2.1 Introduction to python IDE – IDLE**
- **1.2.2 Setting Up Your Environment**
- **1.2.3 Installation of Python and Anaconda Navigator**
- **1.2.4 Quick Tour of Jupyter Notebook**
- **1.2.5 Python vs. IPython**
- **1.2.6 Online compilation support**
- **1.2.7 Running python script using command prompt**

## 1.2 Python Installation Guide

Before you start, you will need Python on your computer.

Installing Python on your computer is the first step to becoming a Python programmer.

Python has two main versions:

- Python 2
- Python 3

However, Python installation differs among different operating systems. The use of Python 3 is highly preferred over Python 2.

## Contd..

For the installation process, go to the official website of Python, i.e., [www.python.org](http://www.python.org). Refer to the current stable version 3.9.4 as of date 13 April 2021. You will get the installer for Python 3.7 or Python 3.9. (at the time of writing). You may even have it with a *32-bit or 64-bit* processor versions.



## 1.2.1 Introduction to python IDE – IDLE

If you have recently installed Python on your computer, you might have seen a new *IDLE* program.

"What is this software doing on my computer?" you might be curious. I did not download that!"

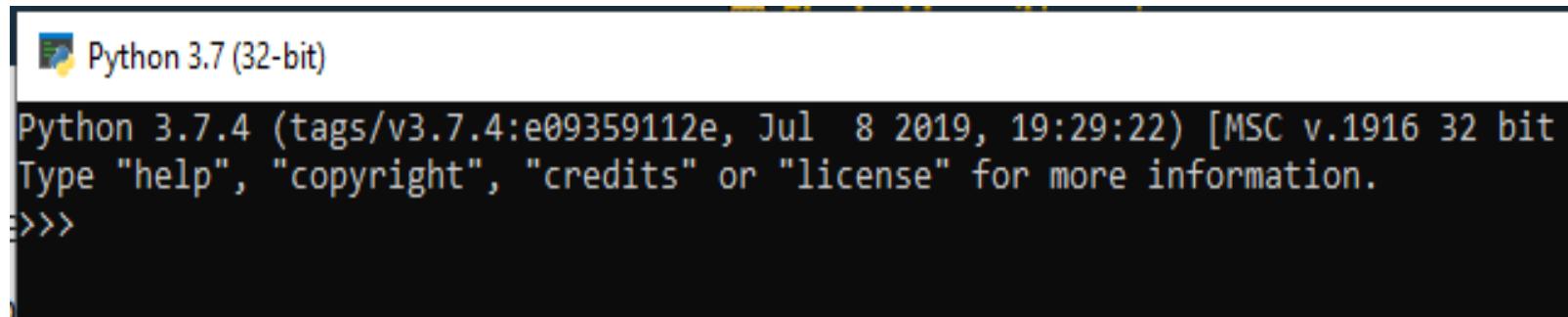
Though you might not have downloaded IDLE on your own, it is included with any Python installation. It is there to help you get acquainted with the language right away.

Any Python installation includes an *Integrated Development and Learning Environment*, abbreviated IDLE or even IDE.

## Contd..

In a graphical user interface (GUI) desktop environment, the installation process puts an icon on the desktop or an object in the desktop menu system that launches Python.

In Windows, for example, there will be a category in the **Start menu** called **Python 3.7**. Under it, a menu item labeled **Python 3.7.4 (32-bit)**.



## Contd..

**Alternate way:** You can also open a terminal window and run the interpreter from the command line.

It is known as **Command Prompt** in Windows. It can be renamed **terminal** in macOS or Linux.

You can type **Windows key+ R** and type **cmd** to open Command Prompt. Then, type **python** to execute python programs.

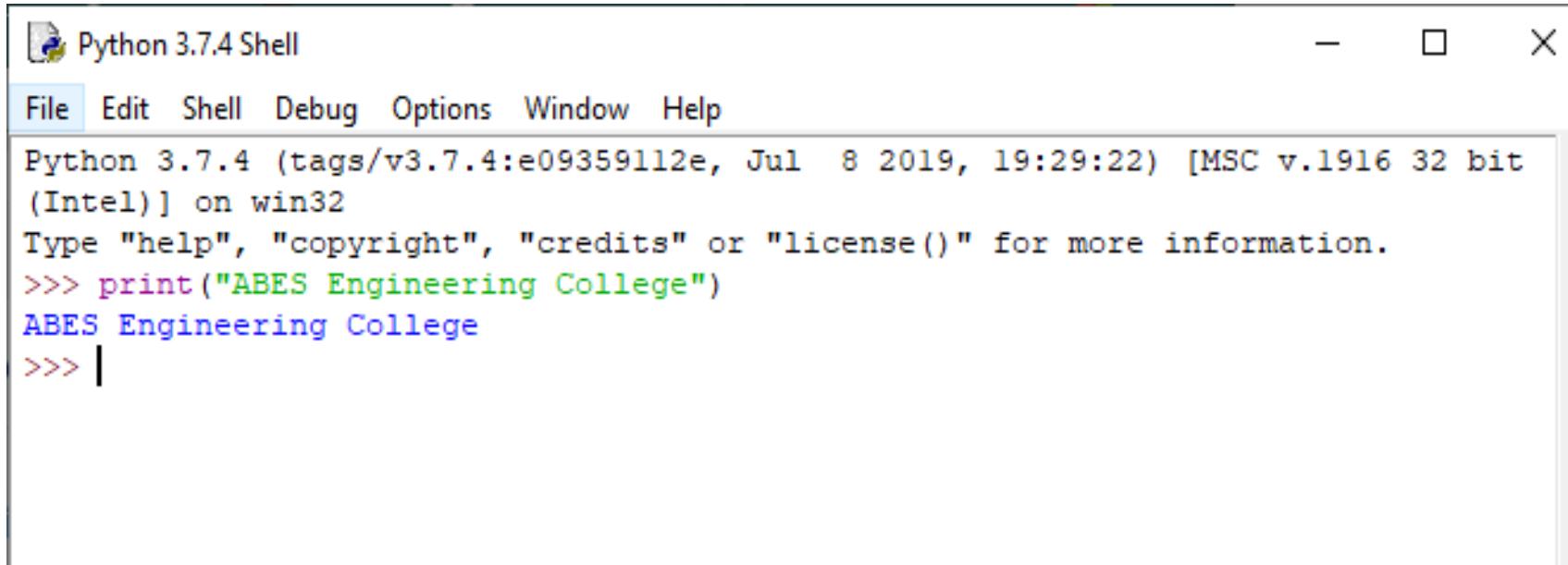
```
C:\WINDOWS\system32\cmd.exe - python
Microsoft Windows [Version 10.0.19041.867]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\Aatif>python
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 19:29:22) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print("ABES Engineering College")
ABES Engineering College
>>>
```

## Contd..

Start working with Python shell.

To **print ()** to display the string "ABES Engineering College" on your computer.  
Enter the command one at a time, and Python returns the results of each command.

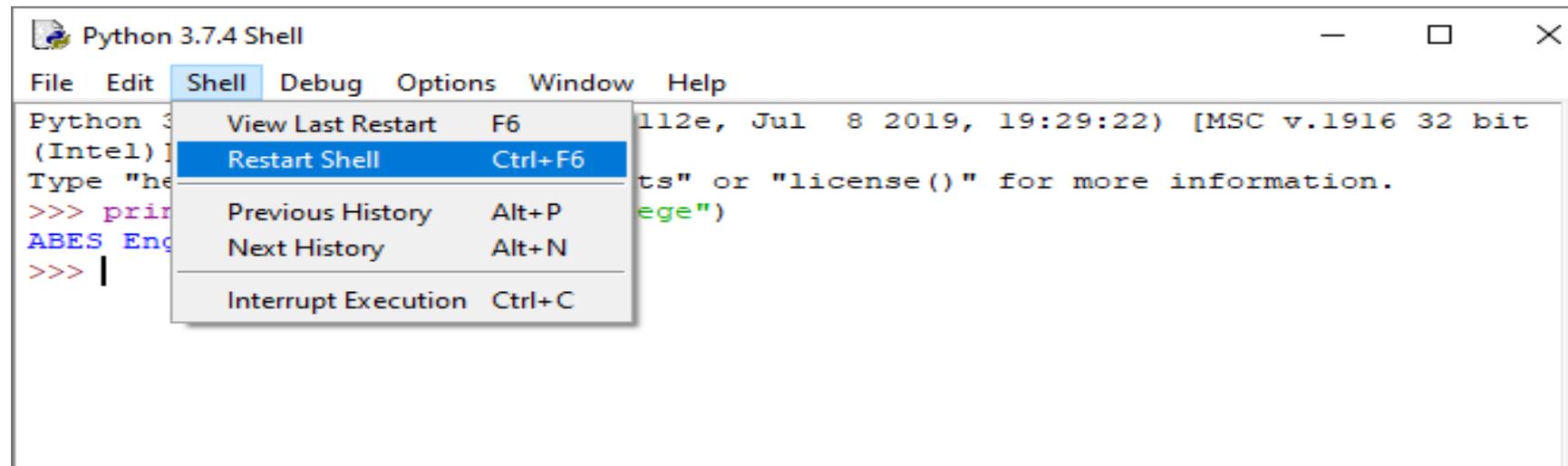


```
Python 3.7.4 Shell
File Edit Shell Debug Options Window Help
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 19:29:22) [MSC v.1916 32 bit
(Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> print("ABES Engineering College")
ABES Engineering College
>>> |
```

## Contd..

From this **menu bar**, you can **restart** the shell. It will behave as if you had launched a new instance of Python IDLE. The shell will forget anything from its former state.

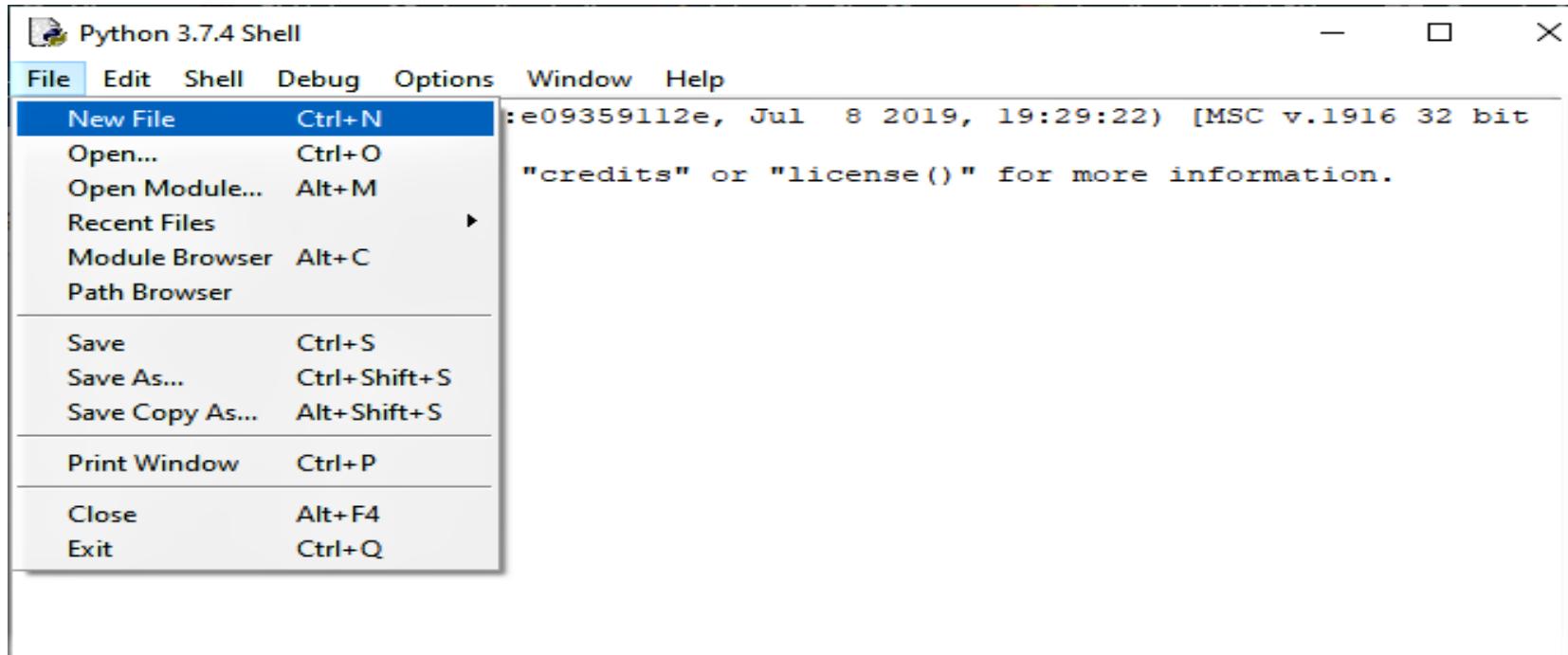
If you want to exit the interpreter, then type **exit ()** and press Enter.



# Python IDLE Editor

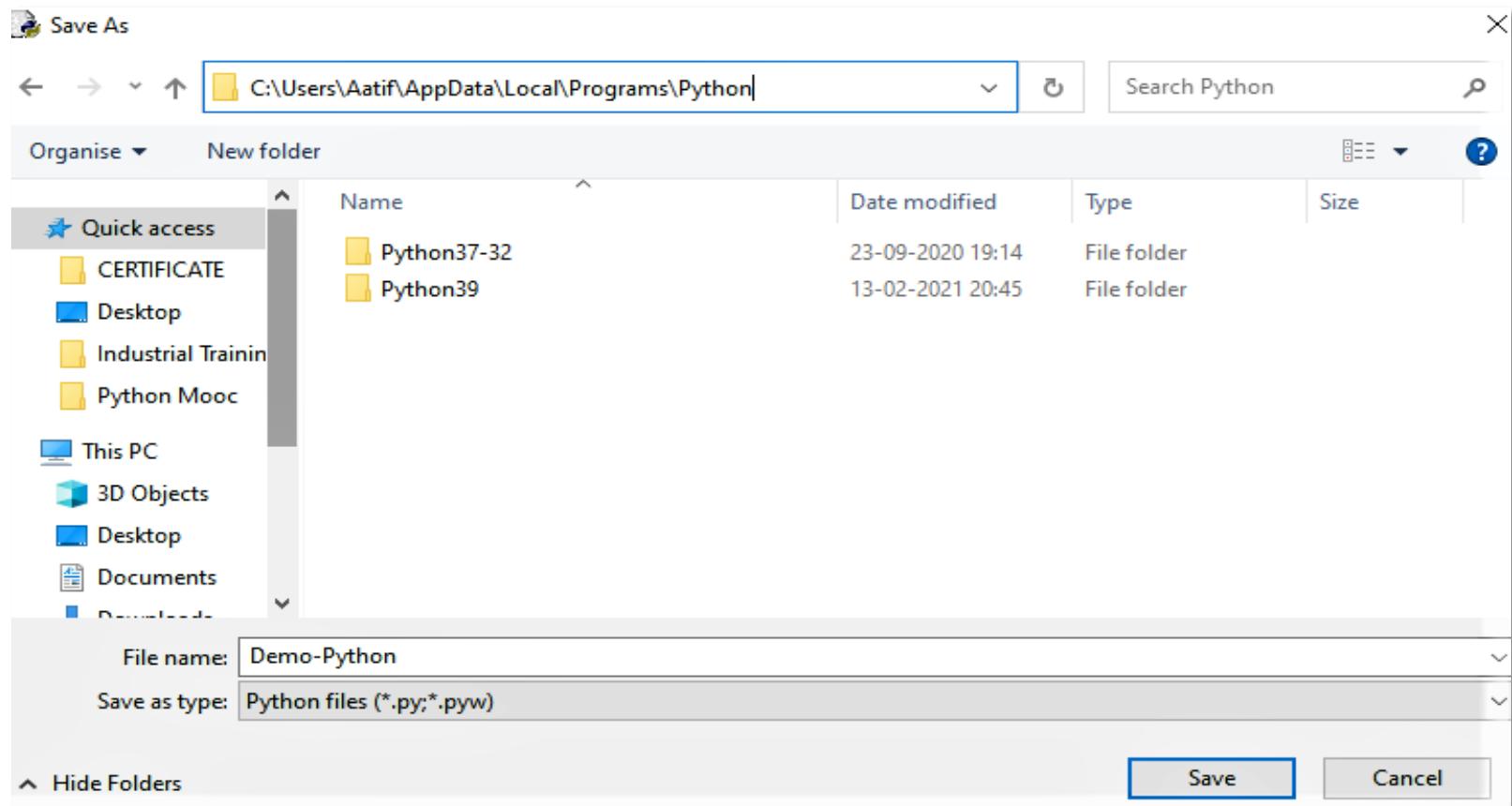
Python **IDLE** includes a full-featured file editor, allowing you to write and run Python programs. The built-in file editor also supplies many tools to speed up your coding workflows, such as code completion and automated indentation.

Select File “**New File**” from the menu bar to begin a new Python file



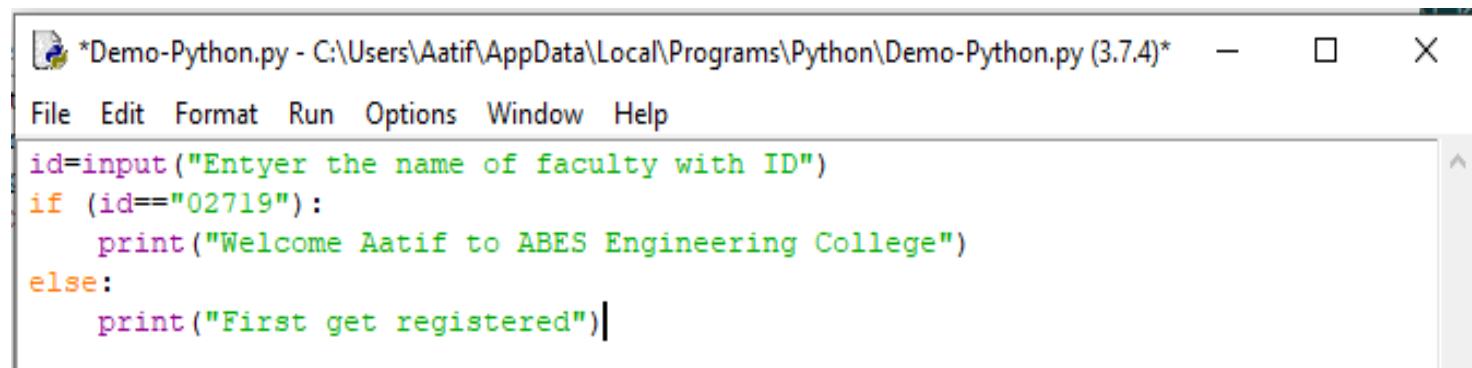
## Contd..

When you are ready to work on a file, click the Edit button. You can save the file as **Demo-Python.py** (.py is an extension to save python scripts files) in the specified default location "C:\Users\Aatif\AppData\Local\Programs\Python."



## Contd..

When you want to run a file, you must first ensure that it has been saved, remember to check for asterisks \* around the filename at the top of the file editor window to see whether the file was correctly saved.



```
*Demo-Python.py - C:\Users\Aatif\AppData\Local\Programs\Python\Demo-Python.py (3.7.4)*
File Edit Format Run Options Window Help
id=input("Enter the name of faculty with ID")
if (id=="02719"):
    print("Welcome Aatif to ABES Engineering College")
else:
    print("First get registered")|
```

But do not panic if you forget! When you want to run an unsaved file in Python IDLE, *it will prompt you to save it*.

Click the **F5 key** on your keyboard to run a file in IDLE. You can also use the menu bar to choose *Run Module*

## 1.2.2 Setting Up Your Environment

The **route**(*path that lists the directories in which the OS (Operating System) looks for executables*) is saved in an environment variable called a string. This variable holds knowledge that the command shell and other programs can use.

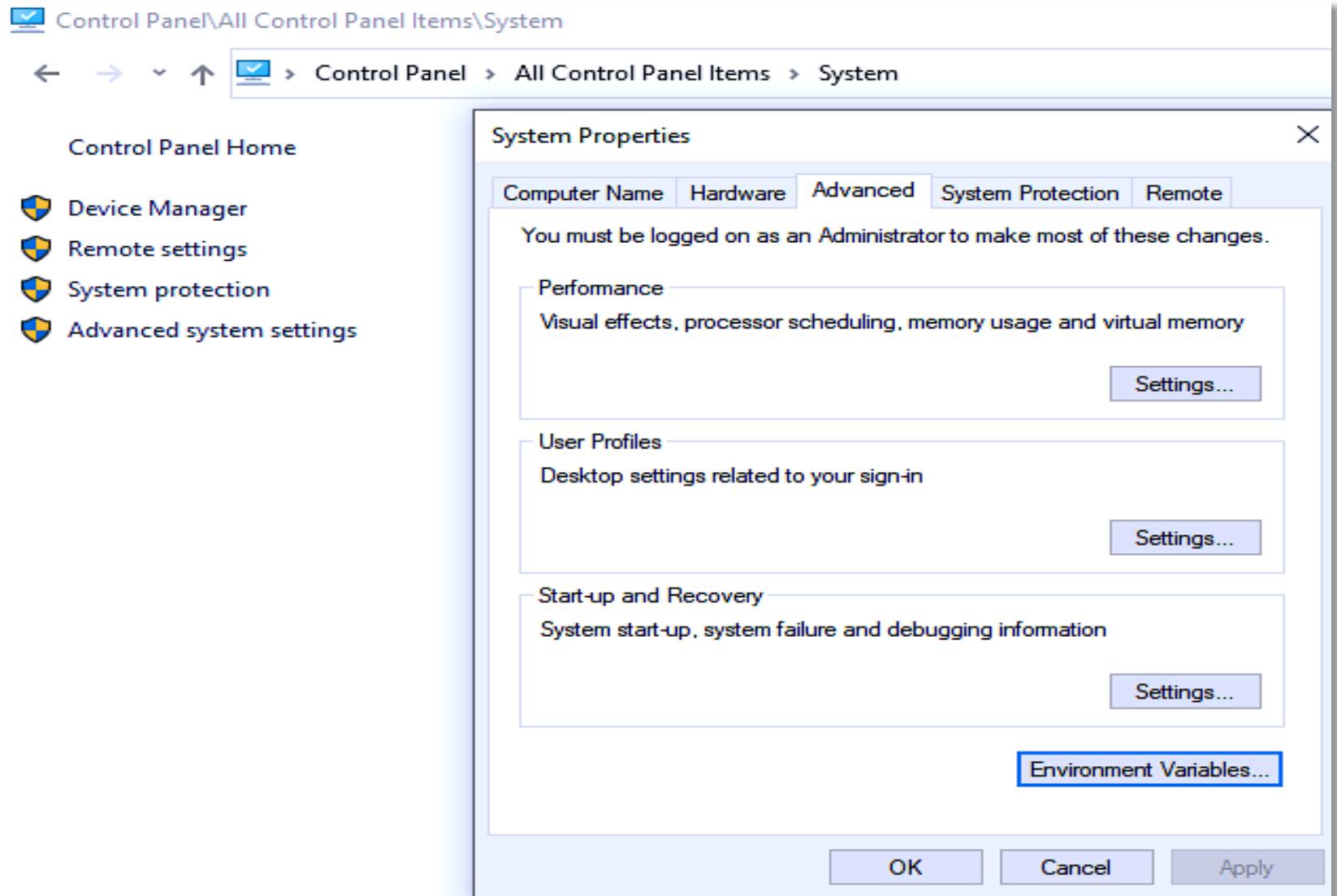
In Unix, the path variable is known as **PATH**, and in Windows, it is known as **Path** (*Unix is case sensitive; Windows is not*).

Following are the steps are taken for setting up the environment:

- *Step 1 – Install Python 3.7 (Latest Version) from python.org.*
- *Step 2-- Add the Python 3.7 Directory to your System Path Environment Variable*

# Contd..

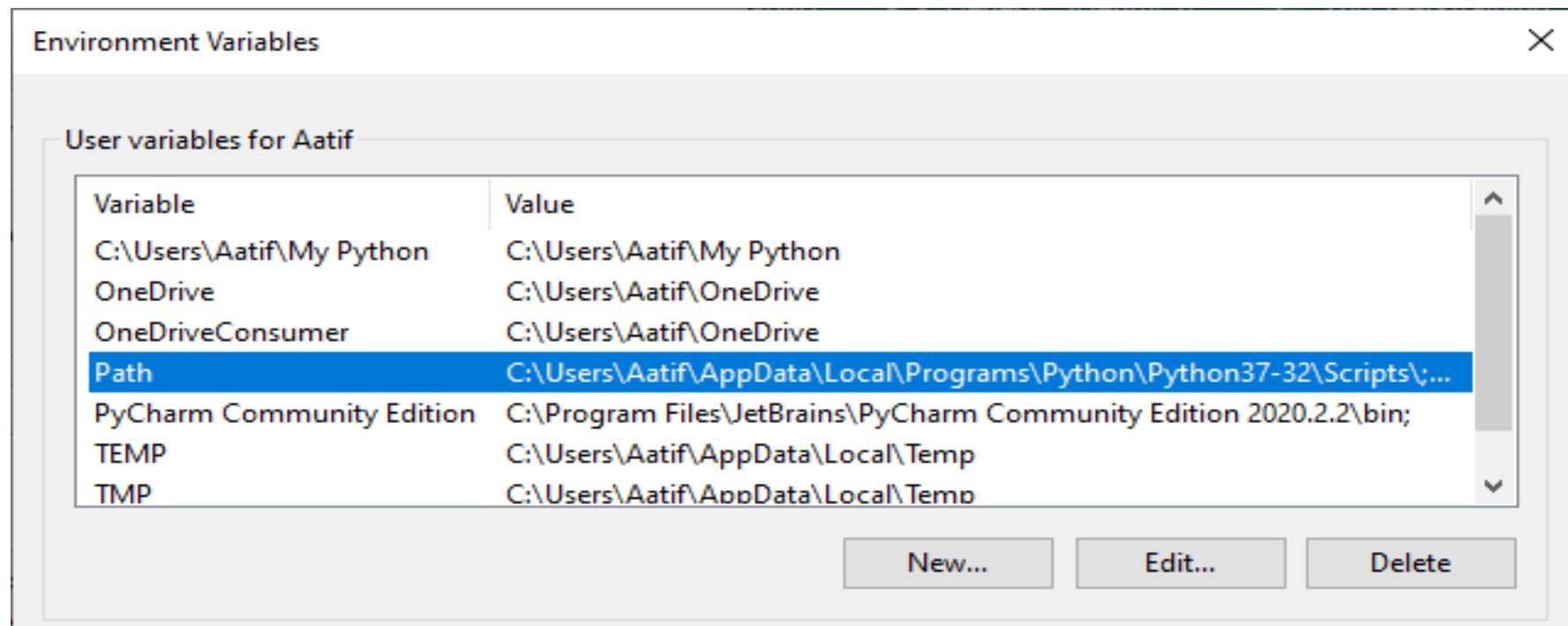
So, navigate to  
*Control Panel* → *System*  
→ *Advanced System Settings*→ *Environment Variables* and choose  
the PATH variable



## Contd..

Add the Python path to the end of the string, this is where the package management software, unit testing tools, and other command line-accessible Python programs can live.

C:\Users\Aatif\AppData\Local\Programs\Python\Python37-32\Scripts\



## 1.2.3 Installation of Anaconda Navigator

Anaconda Navigator is a desktop graphical user interface that comes with Anaconda, which allows you to open programs and control conda packages, environments, and networks without needing to use a command-line interface.

For the installation process, go to the official website of anaconda navigator  
<https://docs.anaconda.com/anaconda/navigator/>.

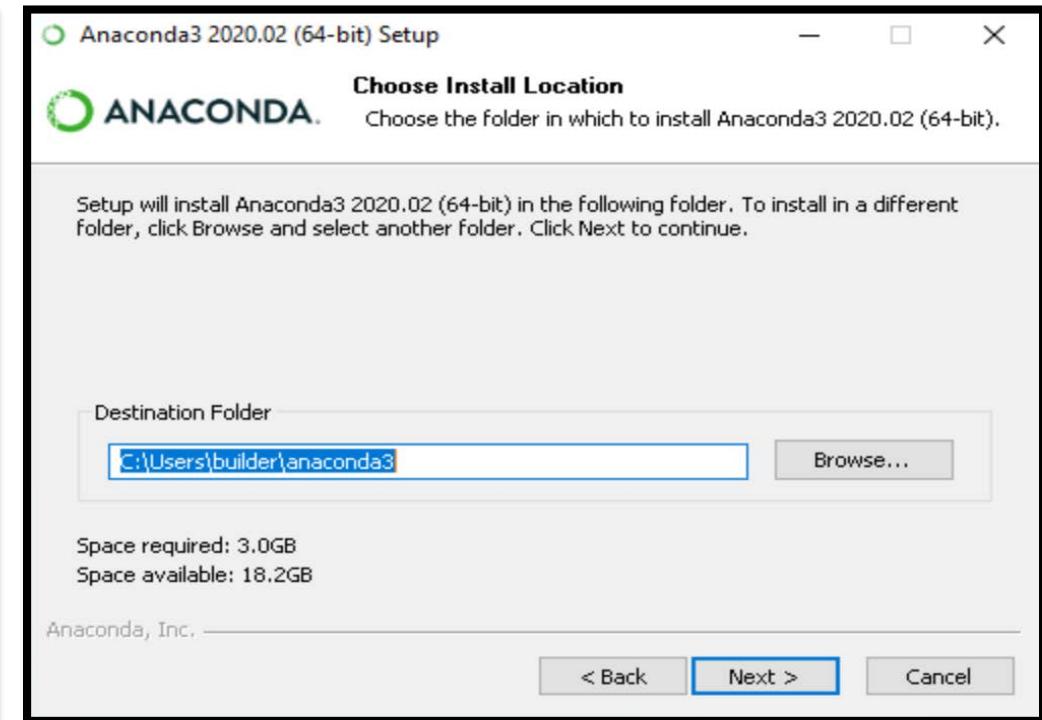
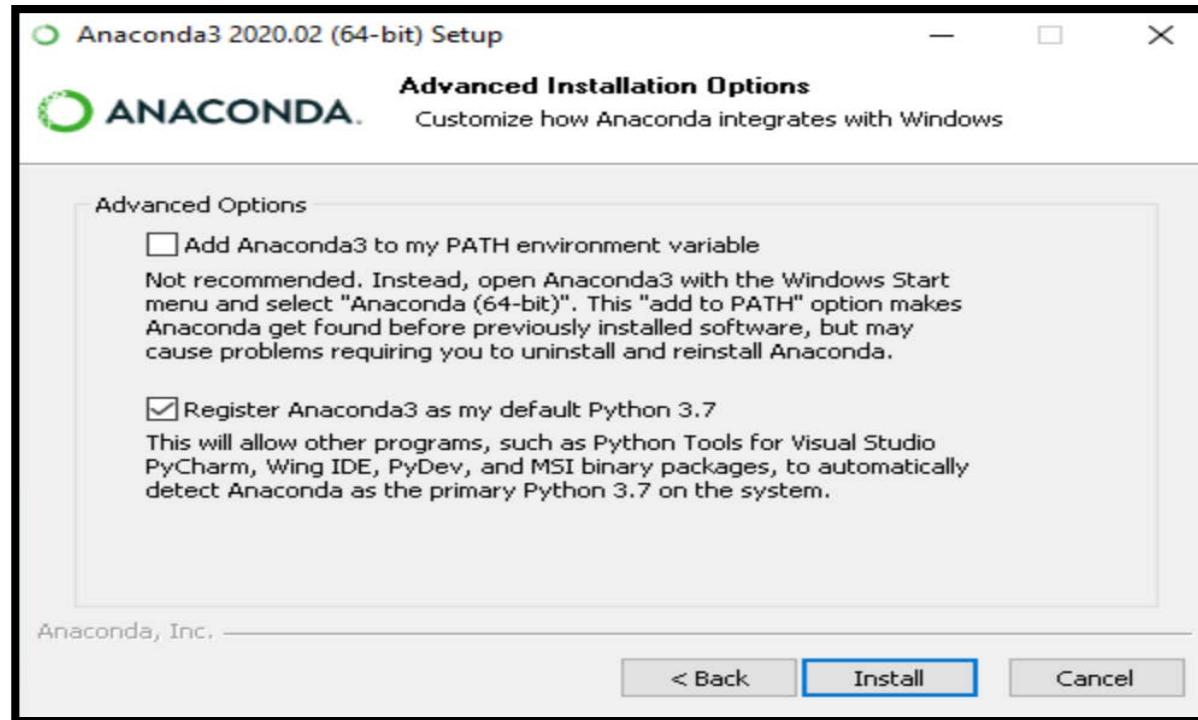
**Latest version:**

*Anaconda3-2020.11-Windows-x86\_64.exe*



# Contd..

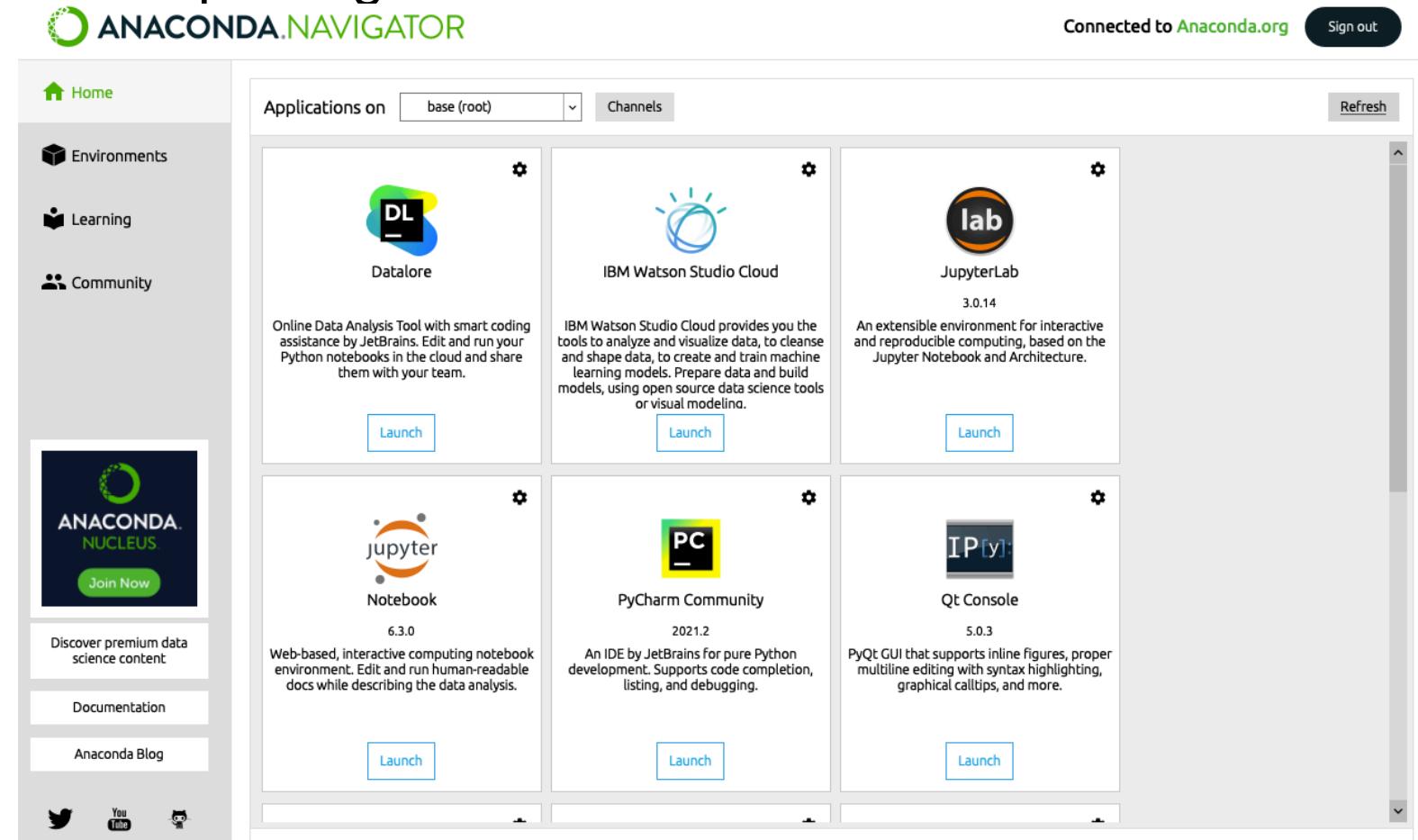
Click on install and Choose destination folder for installation.



## 1.2.4 Quick Tour of Jupyter Notebook

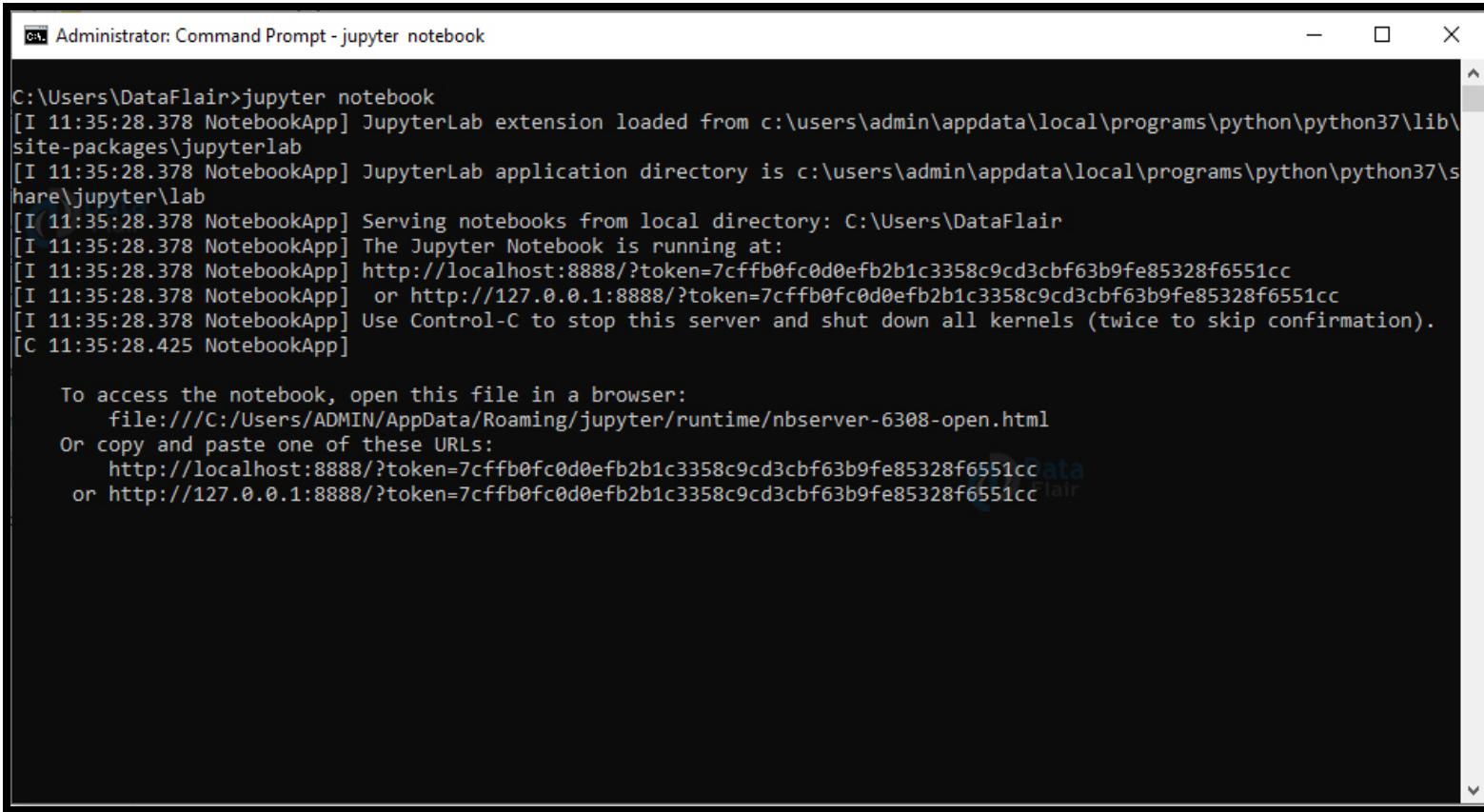
In an earlier topic, we have seen the installation of the Anaconda package. Jupyter comes by default with this package

The *Jupyter Notebook* is a fantastic platform for creating basic and advanced level programs.



## Contd..

**Alternate way:** If you type '**Jupyter notebook**' into your command prompt, it will open the Jupyter dashboard for you



```
C:\Users\DataFlair>jupyter notebook
[I 11:35:28.378 NotebookApp] JupyterLab extension loaded from c:\users\admin\appdata\local\programs\python\python37\lib\site-packages\jupyterlab
[I 11:35:28.378 NotebookApp] JupyterLab application directory is c:\users\admin\appdata\local\programs\python\python37\share\jupyter\lab
[I 11:35:28.378 NotebookApp] Serving notebooks from local directory: C:\Users\DataFlair
[I 11:35:28.378 NotebookApp] The Jupyter Notebook is running at:
[I 11:35:28.378 NotebookApp] http://localhost:8888/?token=7cffb0fc0d0efb2b1c3358c9cd3cbf63b9fe85328f6551cc
[I 11:35:28.378 NotebookApp] or http://127.0.0.1:8888/?token=7cffb0fc0d0efb2b1c3358c9cd3cbf63b9fe85328f6551cc
[I 11:35:28.378 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 11:35:28.425 NotebookApp]

To access the notebook, open this file in a browser:
  file:///C:/Users/ADMIN/AppData/Roaming/jupyter/runtime/nbserver-6308-open.html
Or copy and paste one of these URLs:
  http://localhost:8888/?token=7cffb0fc0d0efb2b1c3358c9cd3cbf63b9fe85328f6551cc
  or http://127.0.0.1:8888/?token=7cffb0fc0d0efb2b1c3358c9cd3cbf63b9fe85328f6551cc
```

## Contd..

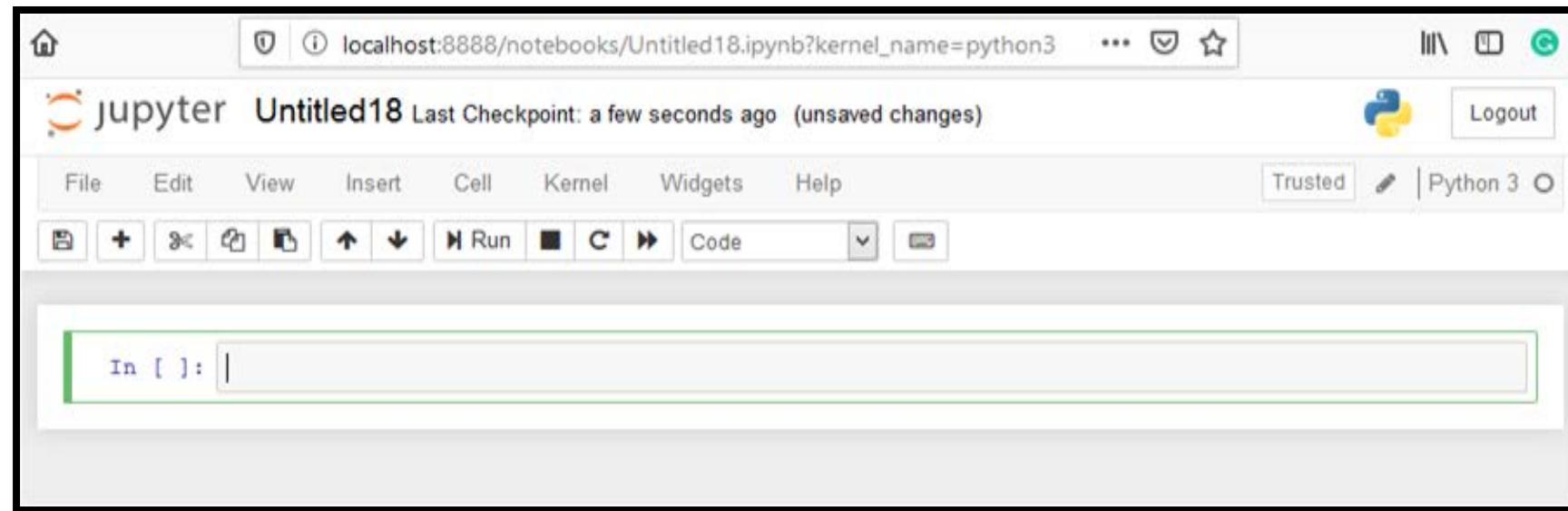
You might have found that the URL for the dashboard is <https://localhost:8888/tree> while Jupyter Notebook is open in your window. The term "localhost" does not refer to a website, but to the fact that the content is served from your own devices.



## Contd..

If you want to create your first file, then you must click on **new** and then **python3**. When you return to the dashboard, you can see the new file **Untitled18.ipynb** with green boundary in cell.

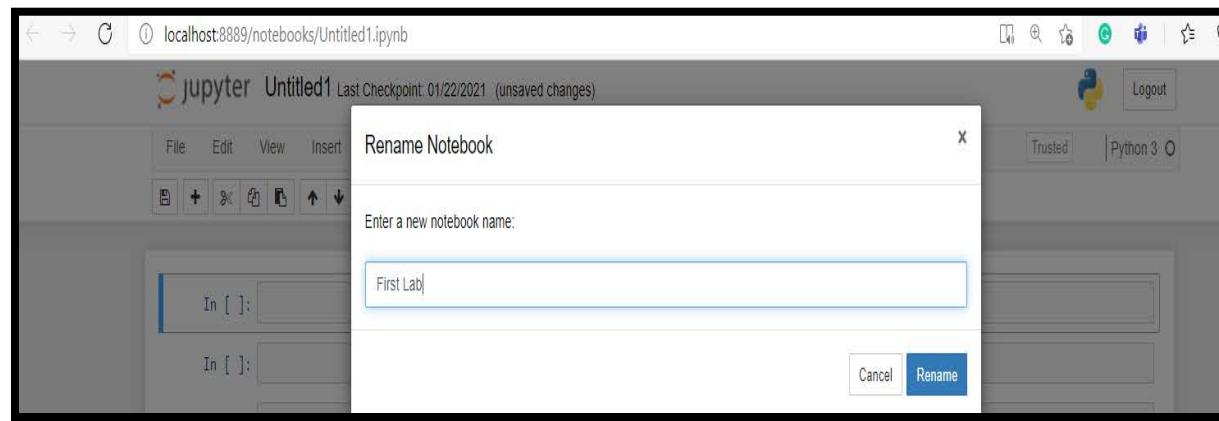
A cell is a container for the text that will be viewed in the notebook or code that the notebook's kernel will execute.



## Contd..

Every *ipynb file* stands for a single notebook, which means that each time you create a new notebook, a new.ipynb file is generated . You should be aware of kernel that are unfamiliar to you.

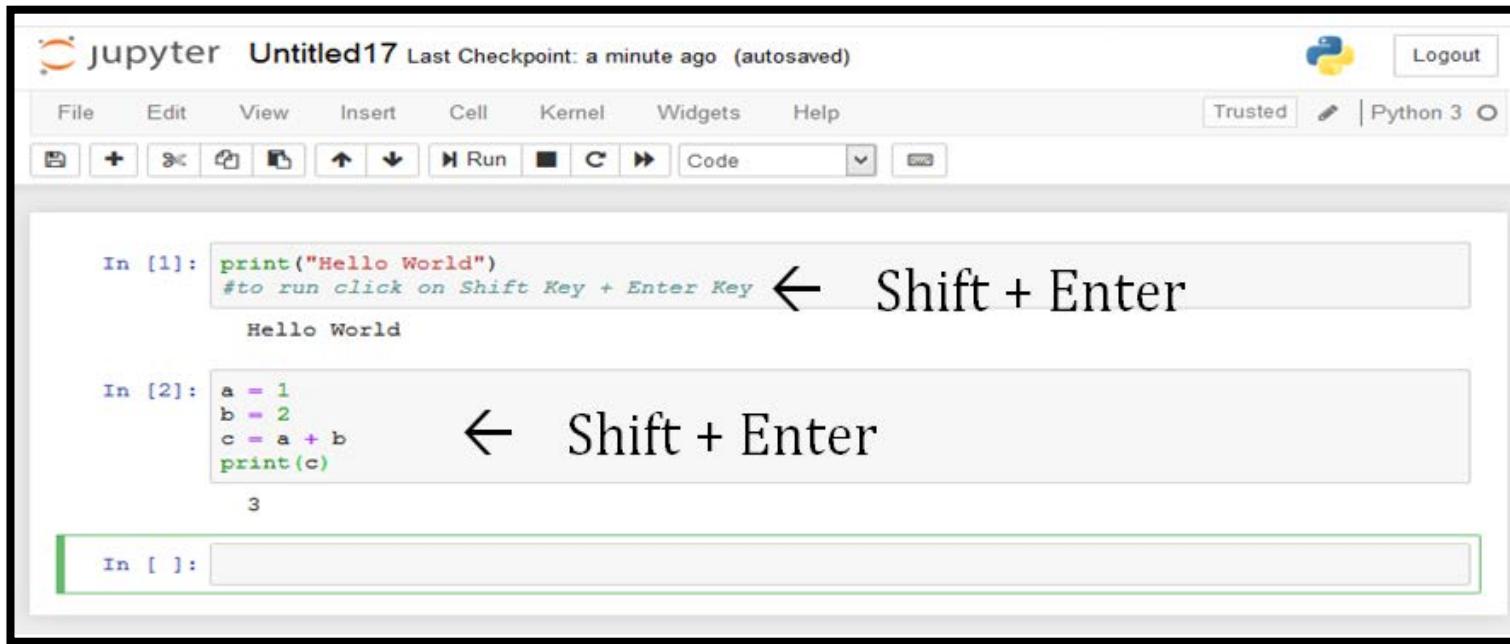
A *kernel* is a kind of "computational engine" that runs the code in a notebook paper.



## Contd..

Shift + Enter is the shortcut command to run your cell.

The green boundary over the cell shows the **editable mode**, and the Blue boundary over the cell shows the **command mode**.



The screenshot shows a Jupyter Notebook interface with two code cells and a new input cell at the bottom.

**Cell 1:**

```
In [1]: print("Hello World")
#to run click on Shift Key + Enter Key
```

← Shift + Enter

Hello World

**Cell 2:**

```
In [2]: a = 1
b = 2
c = a + b
print(c)
```

← Shift + Enter

3

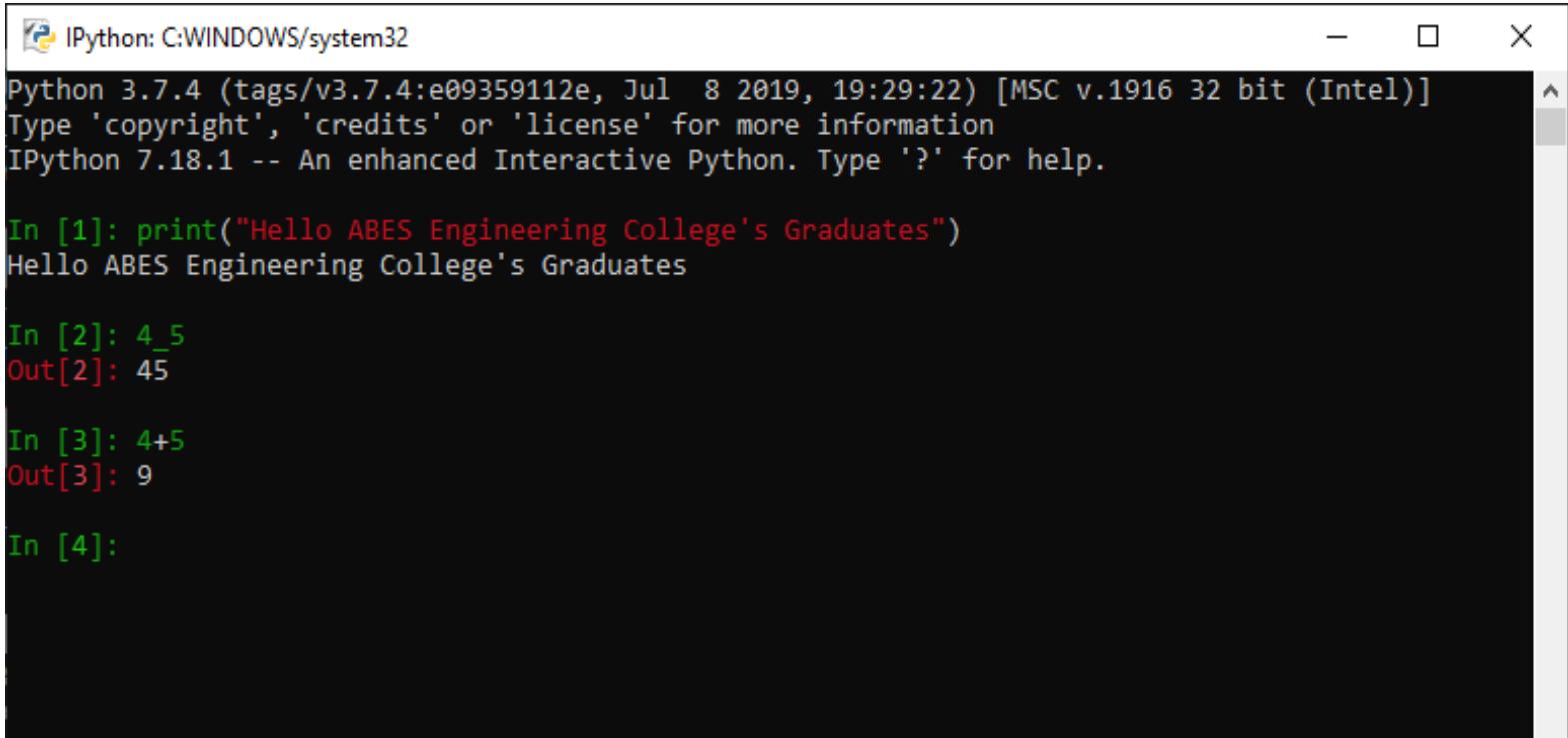
**New Input Cell:**

```
In [ ]:
```

## 1.2.5 Python vs. IPython

IPython's interactive shell is known as Ipython.

IPython is a Python graphical command-line terminal founded by Fernando Perez in 2001. IPython supplies an improved *read-eval-print loop* (REPL) environment that is particularly well suited to scientific computing.



```
IPython: C:\WINDOWS\system32
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 19:29:22) [MSC v.1916 32 bit (Intel)]
Type 'copyright', 'credits' or 'license' for more information
IPython 7.18.1 -- An enhanced Interactive Python. Type '?' for help.

In [1]: print("Hello ABES Engineering College's Graduates")
Hello ABES Engineering College's Graduates

In [2]: 4_5
Out[2]: 45

In [3]: 4+5
Out[3]: 9

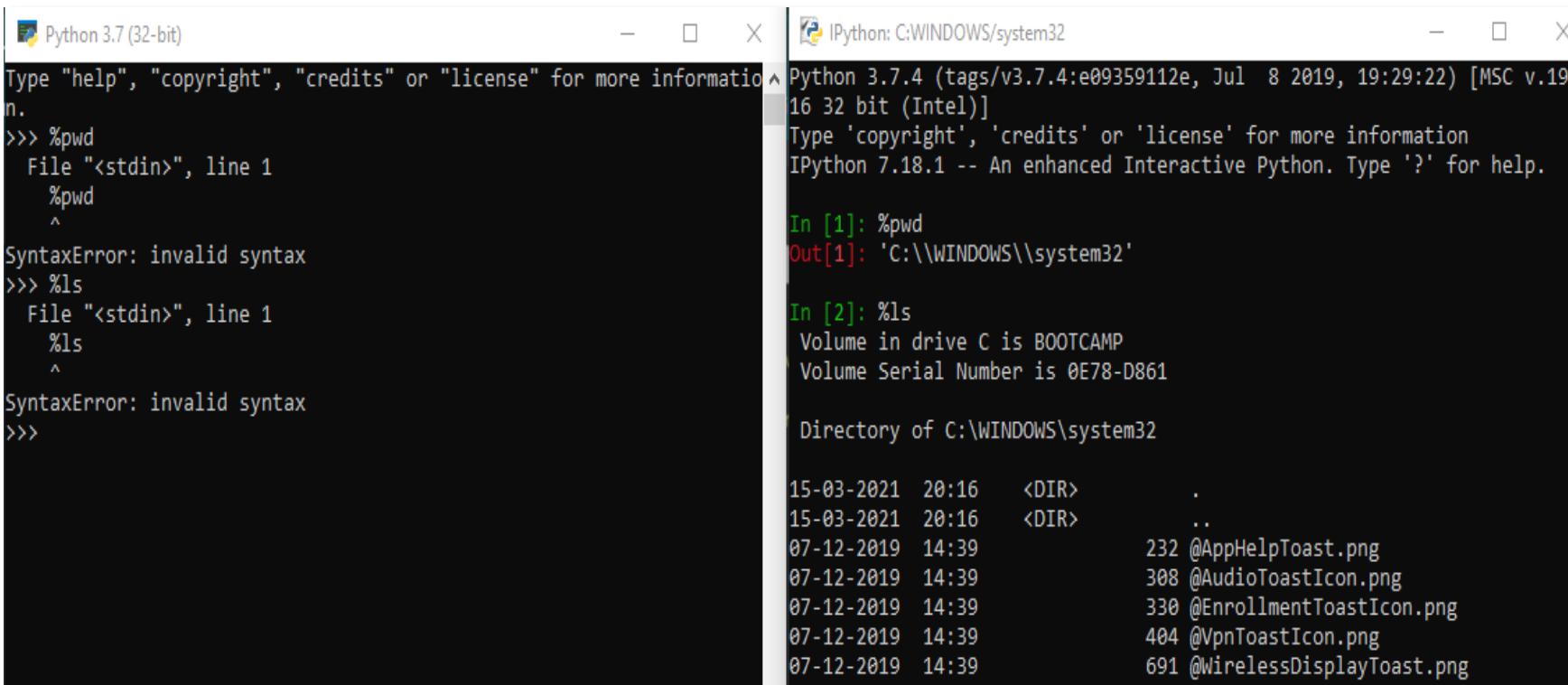
In [4]:
```

# Contd..

**I**Python is interactive, and it gives some relaxation to the eyes of coders by introducing some colors. *Some useful commands are not present with the existing Python idle.*

The following are the valuable commands:

- **%pwd**
- **%ls**
- **%History**



The image shows two side-by-side screenshots of IPython shells. The left shell, titled 'Python 3.7 (32-bit)', displays syntax errors for '%pwd' and '%ls'. The right shell, titled 'IPython: C:\WINDOWS\system32', shows the correct execution of these commands, displaying the current directory and a file listing.

**Left Shell (Python 3.7 (32-bit)):**

```
Type "help", "copyright", "credits" or "license" for more information
n.
>>> %pwd
File "<stdin>", line 1
    %pwd
    ^
SyntaxError: invalid syntax
>>> %ls
File "<stdin>", line 1
    %ls
    ^
SyntaxError: invalid syntax
>>>
```

**Right Shell (IPython: C:\WINDOWS\system32):**

```
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 19:29:22) [MSC v.1916 32 bit (Intel)]
Type 'copyright', 'credits' or 'license' for more information
IPython 7.18.1 -- An enhanced Interactive Python. Type '?' for help.

In [1]: %pwd
Out[1]: 'C:\\WINDOWS\\system32'

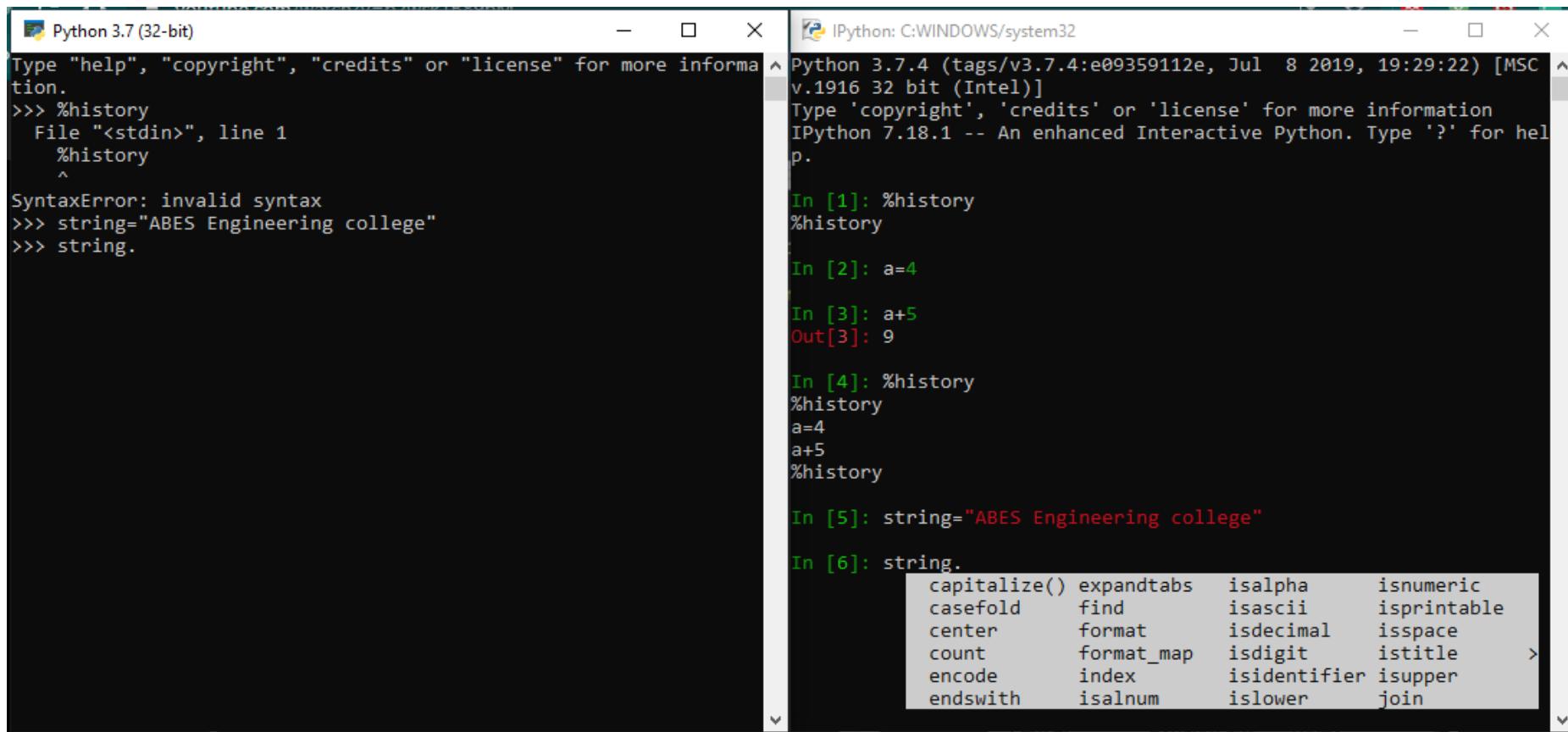
In [2]: %ls
Volume in drive C is BOOTCAMP
Volume Serial Number is 0E78-D861

Directory of C:\\WINDOWS\\system32

15-03-2021  20:16    <DIR>        .
15-03-2021  20:16    <DIR>        ..
07-12-2019  14:39          232 @AppHelpToast.png
07-12-2019  14:39          308 @AudioToastIcon.png
07-12-2019  14:39          330 @EnrollmentToastIcon.png
07-12-2019  14:39          404 @VpnToastIcon.png
07-12-2019  14:39          691 @WirelessDisplayToast.png
```

# Contd..

We can also check **methods** associated with data structures by pressing the tab over the keyboard.



The image shows two side-by-side Python environments demonstrating method completion:

- Python 3.7 (32-bit) Window:** Shows a standard command-line interface. The user types "string." followed by a tab key, which triggers an auto-completion dropdown listing methods like capitalize(), expandtabs(), isalpha(), isnumeric(), etc.
- IPython: C:\WINDOWS\system32 Window:** Shows an enhanced interactive Python environment. The user types "string." followed by a tab key, which triggers an auto-completion dropdown listing methods like capitalize(), expandtabs(), isalpha(), isnumeric(), etc. This window also displays previous commands and their outputs, such as In [1]: %history, In [2]: a=4, In [3]: a+5, Out[3]: 9, and In [4]: %history.

## 1.2.6 Online compilation support

Assume your machine lacks the necessary resources to install, but you need to learn Python or run code to try something.

What if you could run Python online in your browser?

That is very great. Isn't that, right?

You will need a browser, which you already have. Using online IDEs saves your time in the configuration process.

# Contd..

w3schools ([https://www.w3schools.com/python/python\\_compiler.asp](https://www.w3schools.com/python/python_compiler.asp))



The screenshot shows a web-based Python compiler interface. At the top, there are navigation icons for Home, Menu, Device, and Dark Mode, followed by a green 'Run »' button. Below the buttons is a code editor window containing the following Python code:

```
print("Hello, World!")

x = "Python is "
y = "awesome"
z = x + y
print(z)
```

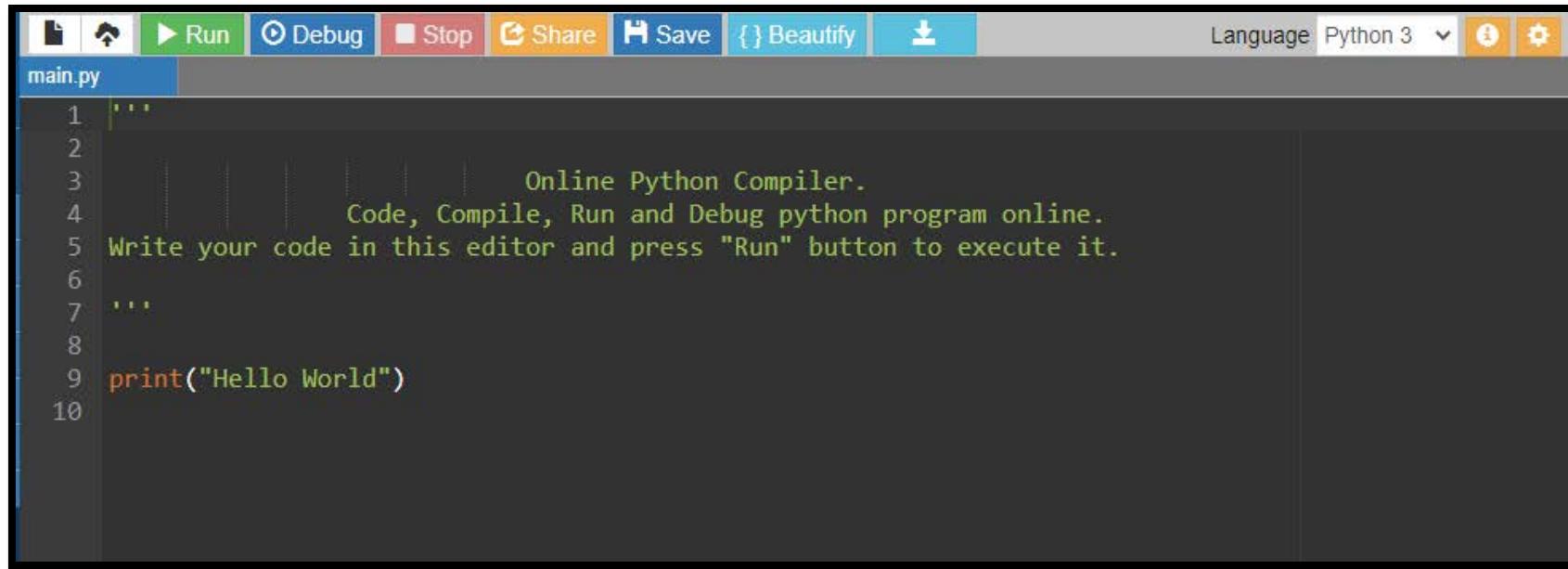
To the right of the code editor is a black output window displaying the results of the execution:

```
Hello, World!
Python is awesome
```

At the bottom left of the interface is a green 'Try it Yourself »' button.

# Contd..

Onlinedb ([https://www.onlinedb.com/online\\_python\\_compiler](https://www.onlinedb.com/online_python_compiler))

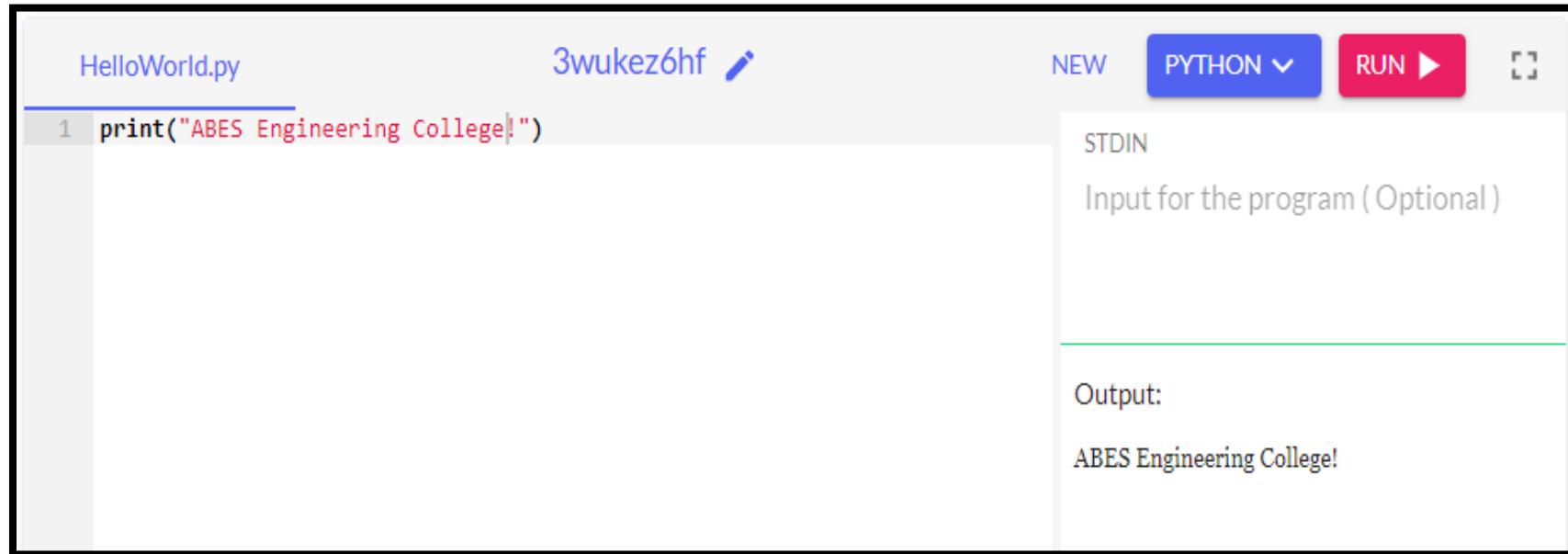


The screenshot shows a web-based Python code editor titled "Online Python Compiler". The toolbar at the top includes buttons for Run, Debug, Stop, Share, Save, Beautify, and a download icon. The language setting is set to Python 3. The code editor window displays a file named "main.py" with the following content:

```
1 """
2
3     Online Python Compiler.
4     Code, Compile, Run and Debug python program online.
5     Write your code in this editor and press "Run" button to execute it.
6
7 """
8
9 print("Hello World")
10
```

# Contd..

One compiler (<https://onecompiler.com/python/3wukez6hf>)



The screenshot shows a Python code editor interface. The file name is 'HelloWorld.py' and the code content is:

```
1 print("ABES Engineering College!")
```

The code is run under the 'PYTHON' environment. The output window shows the result of the print statement:

Output:  
ABES Engineering College!

# Review Questions

- Which version of Python is currently up to date?
  - Python1
  - Python2
  - Python4
  - Python3
  
- The version of Python (if any) that comes pre-installed on your operating system is called \_\_.
  - Onboard Python
  - Monty Python
  - Easy Python
  - System Python

# Review Questions

- In a Python context, the acronym *IDLE* stands for:
  - Integrated Development and Learning Environment
  - Interpretive Dance Lessons
  - Interstellar Dust Laser Explorer
  - None of the above
  
- When you see >>> inside IDLE, it means that:
  - An error has occurred
  - Your computer is having an existential crisis
  - Python is waiting for you to give it some instructions
  - Python is upset

# Session Plan - Day 4

## 1.3 Basics of Python

**1.3.1 Python keywords**

**1.3.2 Python Statement and Comments**

**1.3.3 Python Literals**

**1.3.4 Data Types**

**1.3.5 Variables**

**1.3.6 type (), dir (), ID command**

# Python keywords

Python keywords are **special reserved words** that have specific **meanings and purposes**.

These reserve words cannot be used as a –

- function name
- variable name
- identifiers

Note - As of python 3.9.2, there are **35 reserved** words in Python.

# Contd..

The list of such keywords is mentioned below –

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

# Python Statement and Comments

## Python Statement –

- ❑ In Python Programming, any executable instruction, that tell the computer to perform a specification action is refer to as **statements**.
- ❑ Program statement can be an
  - input-output statements,
  - arithmetic statements,
  - control statements,
  - simple assignment statements
  - and any other statements
  - it can also includes comments.

# Python Literals

Literals are the **type of data** that is used to store in a **variable or constant**.

Types of python literals:

- ❑ String literals
- ❑ Numeric Literals
  - ❑ Integer Literals
  - ❑ Float Literals
  - ❑ Complex Number Literals
- ❑ Boolean Literals
- ❑ Special literals
- ❑ Literal Collections

# String literals

When set of character are enclosed in quotes ( single quotes or double quotes) then it formed a string literals.

Example -

'abes'

'rate\_of\_interest'

'123',

'12.5'

"ABESEC"

"Simple\_interest"

"A1"

"456 "

In Python we can also create multi-line literals by using '\'.

Example -

'ABES Engineering College \  
NH-24 Delhi Hapur Bypass \  
Near Crossing Republic'

"ABES Engineering College \  
NH-24 Delhi Hapur Bypass \  
Near Crossing Republic"

# Numeric Literals

Numeric literals are of multiple types based on the number type. The types of numeric literals are **integer, float (decimal numbers), and complex numbers.**

**Integer Literals** - They can be either positive or negative.

Example –

**12, 23000000, -45, 0 , -23987**

**Float Literals** - These are basically real numbers that consist of both integer as well as fractional parts.

Example -

**-12.45, 12.90, 100.0**

# Numeric Literals

**Complex Number Literals** - The numerals will be in the form of  $a + bj$ , where 'a' is the real part and 'b' is the complex part.

Python allows us to specify complex numbers like any other variable.

Example -

10j , 1 + 0j , 10 + 2j, 12 - 5j

# Boolean Literals , Special literals

**Boolean Literals** - True or False are the values to be used as the Boolean values.

Example -

**True, False**

In Python, True represents the Non-zero value and False represents the value as Zero.

**Special literals** - Python has a special literal named None.

Note - None is used to signify the NULL value.

# Literal Collections

**List Literals** – List is a set of values of different types. The values are separated by comma (,) and enclosed within square brackets( [ ]).

```
[ 1, 23, 23.4, 100 ]
```

Example –

```
[ 'Blue', 'red', 123, 23.5 ]
```

**Tuple literals** – A tuple is a set of values of different types. The values are separated by comma(,) and enclosed within parentheses “ ( ) ”. It is immutable.

Example -

```
( 1, 23, 23.4, 100 )
```

```
( 'Blue', 'red', 123, 23.5 )
```

# Literal Collections

**Dictionary literals** – It is in form of key-value pair. It is enclosed by curly-braces “{ }” and each key-value pair is separated by commas (,) .

**Example** - { 'name' : 'BOB' , 'age' : 24 , 'marks' : 59.4 }

**Set literals** – Set is a collection of values of different types. The values are separated by comma (,) and enclosed within curly-braces “{ }”. It is unordered and contains only unique value.

**Example** -

{ 1, 23, 45, 56, 67 }

{ 'Ram', 'Rajesh', 12, 34.5 }

# Variables

Variable are the names given by the users to the memory locations to store the data values.

In Python, variable need not to be declared or defined in advances, as we do in many other programming language.

To create a variable, we just assign it a value and start using it.

Example –

Here ‘**a**’, ‘**name**’ and ‘**Marks**’ are variable which refer an integer value 23, a string ‘Ram’ and float Value 23.5.

**a = 23**

**name = 'Ram'**

**Marks = 23.5**

# Variables

## Rules for variable name –

- Variables can be named with an alpha-numeric combination, started with an alphabet or underscore.
- Variable name can't start with digit.
- Multi- word space separated name can't be used as a variable name.
- The reserved words(keywords) cannot be used naming the variable.

### Valid Variable Names

**Name, num1,  
rate\_of\_interest,  
\_abc, marks**

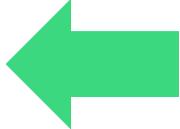
### In-Valid Variable Names

**for, 123b, rate of  
interest, marks-math**

# Can you answer these questions?

1. Select all Valid variable names -

1) age



2) \_age



3) -age

4) age\_\*

5) Item-Number-1

# type() command

**type () command** helps in finding the type of the specific declared variable or a value.

Example –

```
a = 23  
type(a)
```

int

```
name = "Ram"  
type(name)
```

str

```
b = 23.5  
type(b)
```

float

```
c = 10+2j  
type(c)
```

complex

# id() command

**id () command** gives the unique id for a given objects / variable / values.

**Note -** The unique id is the memory address and will be different each time when you run for variable or values.

Example –

Unique Identity

```
a = 23  
id(a)
```

140707530484192

```
name = "Ram"  
id(name)
```

3089171110640

# dir() command

**dir () command** is a vital function that returns all the properties and methods associated with given objects.

## Example –

```
#dir() command
x = 2
dir(x)
```

Note – Detailed description will be explained in OOPs

### Output:

<code>'__abs__', '__add__', '__and__', '__bool__', '__ceil__', '__class__', '__delattr__', '__dir__', '__divmod__', '__doc__', '__eq__', '__float__', '__floor__', '__truediv__', '__trunc__', '__xor__', 'as_integer_ratio', 'bit_length', 'conjugate', '__floordiv__'</code>	<code>'__format__', '__ge__', '__getattribute__', '__getnewargs__', '__gt__', '__hash__', '__index__', '__init__', '__init_subclass__', '__int__', '__invert__', '__le__', '__lshift__', 'denominator', 'from_bytes', 'imag', '__lt__', '__mod__', '__mul__', '__ne__', '__neg__', '__new__', '__or__', '__pos__', '__pow__', '__radd__', '__rand__', '</code>	<code>'__rdivmod__', '__reduce__', '__reduce_ex__', '__repr__', 'numerator', 'real', 'to_bytes']', '__rfloordiv__', '__rlshift__', '__rmod__', '__rmul__', '__ror__', '__round__', '__rpow__', '__rrshift__', '__rshift__', '__rsub__', '__rtruediv__', '__rxor__', '__setattr__', '__sizeof__', '__str__', '__sub__', '__subclasshook__'</code>
--	--	--

# Session Plan - Day 5

## 1.3 Basics of Python

**1.3.7 Type conversion: implicit and explicit**

**1.3.8 Basic I/O Operations: input (), print ()**

# Type conversion

The process of converting the value of one data type (e.g. integer, string, float, etc.) to another data type is called type conversion.

## Example -

12.5 → 12	( float to integer )
'123' → 123	( string to integer )
12 → 12.0	( integer to float )

Python has two types of type conversion –

- Implicit Type Conversion
- Explicit Type Conversion

# Type conversion : Implicit Type Conversion

**Implicit Type Conversion** – In this Python interpreter itself converts one type of data to another data type as per the performed operation.

This type conversion happens automatically without any user intervention.

Example –

```
a = 12
b = 2.5
c = a+b
type(c)
```

float

```
a = 13
b = 3
c = a/b
type(c)
```

float

**Note** - Python promotes the conversion of the lower data type (integer) to the higher data type (float) to avoid data loss.

# Type conversion : Explicit Type Conversion

**Explicit Type Conversion** – In this, user converts the data type of variable of value to needed data type.

Example –

```
a = 13
b = 3
c = int(a/b)
type(c)
```

int

Note – There should be valid Numbers while converting any string to Numbers.

**int() → for Integer**  
**float() → for Float**  
**str() → for string**  
**Complex → for complex**

# Can you answer these questions?

1. What will be the output of the following -

- a) 12 (20+0j) 123.45
- b) 12 20 123.45
- c) Error

```
a = 12.4
b = 20
c = '123.45'
print(int(a), complex(b), float(c))
```

# Basic I/O Operations

In python, various built-in functions are present.

The two important standard input-output functions in python are:

- `input()` : to take the input from the user
- `print()` : to show the output on the console

# Basic I/O Operations : input()

## Syntax –

```
input(prompt='')
```

**Prompt** - A String, representing a default message before the input.

## Example –

```
a = input("Enter any number")
```

Enter any number

**Note** – It return value as a string. So you need to typecast it.

# Basic I/O Operations : input()

## Syntax –

```
input(prompt=' ')
```

**Prompt** - A String, representing a default message before the input.

## Example –

```
a = input("Enter any number")
```

Enter any number

# Basic I/O Operations : input()

When we take input using input() function, it return value in string data type.

**Example –**

```
a = input("Enter first Value -> ")
type(a)
```

Enter first Value -> 23

str

As we can see in the above example type of “a” is string

```
a = int(input("Enter first Value -> "))
type(a)
```

Enter first Value -> 23

int

# Basic I/O Operations : print()

**print ()** is a built-in standard function used to print the output to the console.

Syntax –

```
print(value, ..., sep=' ', end='\n')
```

- value – Can be of any literals, variable, expression, statements
- sep - (optional), Specify how to separate the values, if there is more than one. Default is ''.
- end - (optional), Specify what to print at the end. Default is '\n'

Note – We can assign any set of character into sep.

# Basic I/O Operations : print()

Example –

```
a=10  
b=23.5  
print(a,b)
```

10 23.5

In the above example value of a and b is separated by space i.e. default

Here separator is `sep='--'`, so both value is separated by '--' as shown in output.

```
a=10  
b=23.5  
print(a,b,sep='--')
```

10--23.5

# Basic I/O Operations : print()

**Example – Write a Python program that takes two integer as input from user and print sum of both.**

```
a = int(input("Enter first Value -> "))
b = int(input("Enter second Value -> "))
c = a+b
print(c)
```

```
Enter first Value -> 10
Enter second Value -> 20
30
```

# Session Plan - Day 6

## 1.3 Basics of Python

### 1.3.9 Operators

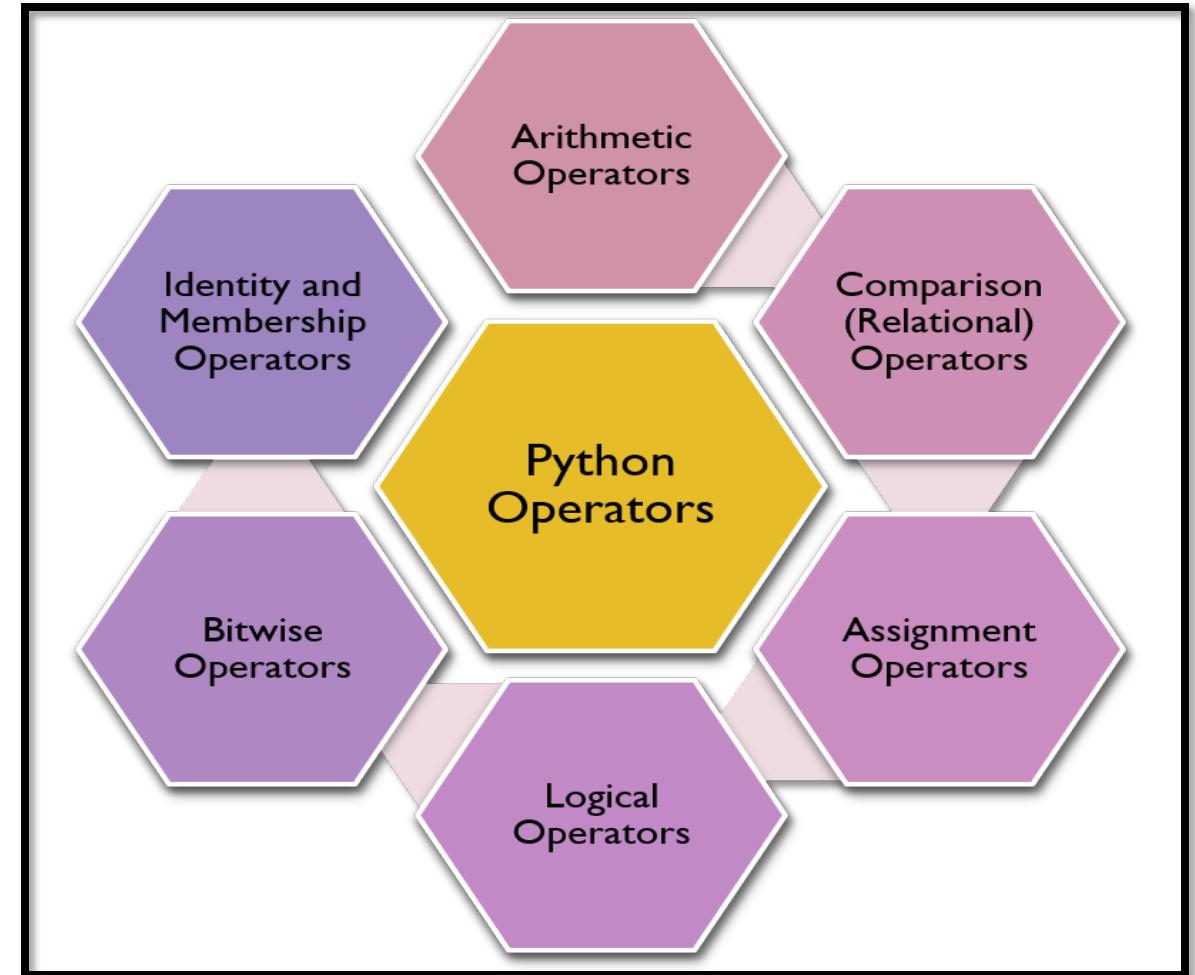
### 1.3.10 Precedence and associativity

### 1.3.11 Python 2 vs Python 3

# Operators

**Operators** are symbol, used to perform mathematical and logical operation.

In python operators are categorized into six categories -



# Arithmetic operators

There are seven arithmetic operators, and these are of:

Operator	Meaning	Example
+	Add two operands or unary plus	$x + y$ +2
-	Subtract right operand from the left or unary minus	$x - y$ -2
*	Multiply two operands	$x * y$
/	Divide left operand by the right one (always results into float)	$x / y$
%	Modulus - remainder of the division of left operand by the right	$x \% y$ (remainder of $x/y$ )
//	Floor division - division that results into whole number adjusted to the left in the number line	$x // y$
**	Exponent - left operand raised to the power of right	$x^{**}y$ ( $x$ to the power $y$ )

# Arithmetic operators

## Example of all mentioned arithmetic operators

```
x = 4
y = 5

print('x + y =',x+y)
print('x - y =',x-y)
print('x * y =',x*y)
print('x / y =',x/y)
print('x // y =',x//y)
print('x ** y =',x**y)
```

```
x + y = 9
x - y = -1
x * y = 20
x / y = 0.8
x // y = 0
x ** y = 1024
```

# Comparison (Relational) Operators

The comparison operators are used for comparisons.

Comparison operators compare two values and evaluate down to a single **Boolean value ( True / False )**.

Operator	Meaning	Example
>	Greater than -> True if left operand is greater than the right	$x > y$
<	Less than -> True if left operand is less than the right	$x < y$
==	Equal to -> True if both operands are equal	$x == y$
!=	Not equal to -> True if operands are not equal	$x != y$
>=	Greater than or equal to -> True if left operand is greater than or equal to the right	$x >= y$
<=	Less than or equal to -> True if left operand is less than or equal to the right	$x <= y$

# Comparison (Relational) Operators

Example of all mentioned comparison operators

It always gives answer either **True or False** depending upon relation.

```
a = 10
b = 20

print(a > b)
print(a < b)
print(a == b)
print(a != b)
print(a >= b)
print(a <= b)
```

```
False
True
False
True
False
True
```

# Comparison (Relational) Operators

## Example –

```
'hello' == 'hello'
```

True

```
'hello' == 'Hello'
```

False

```
'dog' != 'cat'
```

True

```
50 == '50'
```

False

```
25 == 25.0
```

True

**Note** - The == and != operators can actually work with values of any data type.

# Bitwise Operators

Bitwise operators act on the bits and performs bit by bit operation on the operands.

Example – Evaluate 2 & 7

**How it works -**

Step 1 – Convert 2 in binary → 0010

Step 2 – Convert 7 in binary → 0111

Step 3 – Perform Bitwise & operation

Step 4 – Convert the result back to decimal

Operator	Meaning
&	Bitwise AND
	Bitwise OR
~	Bitwise NOT
^	Bitwise XOR
>>	Bitwise right shift
<<	Bitwise left shift

# Bitwise Operators – Cont..

## Truth Table

A	B	A & B	A   B	A ^ B
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

## Bitwise NOT

A	$\sim A$
0	1
1	0

# Bitwise Operators – Bitwise AND (&)

## Example –

```
a = 9  
b = 3  
print(a & b)
```

1

## Explanation –

Step 1 – Convert 9 in binary → 1001

Step 2 – Convert 3 in binary → 0011

Step 3 –

$$\begin{array}{r} 1001 \\ 0011 \\ \hline 0001 \end{array}$$

Step 4 – Convert the result (0001) back to decimal → 1

# Bitwise Operators – Bitwise OR( | )

## Example –

```
a = 9  
b = 3  
print(a | b)
```

11

## Explanation –

Step 1 – Convert 9 in binary → 1001

Step 2 – Convert 3 in binary → 0011

Step 3 –

$$\begin{array}{r} 1001 \\ 0011 \\ \hline 1011 \end{array}$$

Step 4 – Convert the result (1011) back to decimal → 11

# Bitwise Operators – Bitwise XOR ( ^ )

## Example –

```
a = 9  
b = 3  
print(a ^ b)
```

10

## Explanation –

Step 1 – Convert 9 in binary → 1001

Step 2 – Convert 3 in binary → 0011

Step 3 –

$$\begin{array}{r} 1001 \\ 0011 \\ \hline 1010 \end{array}$$

Step 4 – Convert the result (1010) back to decimal → 10

# Bitwise Operators – Bitwise Left shift ( << )

## Example –

```
print(9<<1)
```

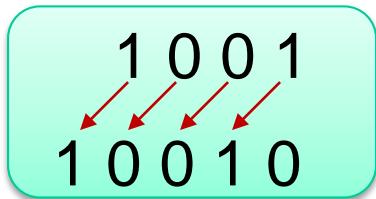
18

## Explanation –

Step 1 – Convert 9 in binary → 1001

Step 2 – Shift 1001 towards left by one position and place Zero at end.

Step 3 – Convert the result (10100) back to decimal → 18



# Bitwise Operators – Bitwise Right Shift ( >> )

## Example –

```
print(9>>1)
```

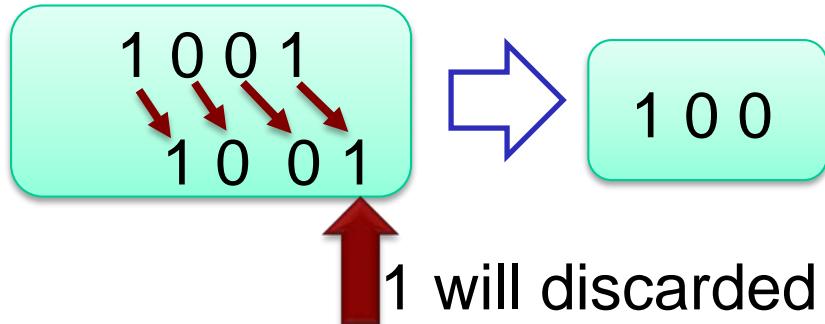
4

## Explanation –

Step 1 – Convert 9 in binary → 1001

Step 2 – Shift 1001 towards right by one position.

Step 3 – Convert the result (100) back to decimal → 4



# Assignment operators

Assignment operators are used to assign the values to the variables.

**a = 5** is a simple assignment operator that assigns the **value 5** on the right to the variable **a on the left.**

There are various compound operators in Python like **a += 5**

Operator	Example	Equivalent 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

# Logical Operators

Logical operators perform Logical AND, Logical OR and Logical NOT operations.

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

**Truth Table –**

x	y	x and y	x or y
T	T	T	T
T	F	F	T
F	T	F	T
F	F	F	F

x	not x
T	F
F	T

# Logical Operators

Example –

```
a=10
b=20
print(a<b and a!=b)
print(a>b or b>a)
print(not a)
```

True  
True  
False

# Identity Operators

**Identity Operators – `is` and `is not`** are the identity operators in Python.

They are used to check if two values (or variables) are located on the same part of the memory or not.

Operator	Meaning
<code>is</code>	Gives <b>True</b> if the operands are identical (refer to the same ID or Memory)
<code>is not</code>	Gives True if the operands are not identical (do not refer to the ID or Memory)

## Example

```
a = 10
b = 10
c = 12
print(a is b)
print(a is c)
print(a is not c)
```

```
True
False
True
```

# Membership Operators

**in** and **not in** are the membership operators in Python.

They are used to test whether a value or variable is found in a sequence (string, list, tuple, set and dictionary) or not.

Operator	Meaning
<b>in</b>	Gives <b>True</b> if value/variable is found in the sequence otherwise <b>False</b>
<b>not in</b>	Gives <b>True</b> if value/variable is not found in the sequence otherwise <b>True</b>

Example

```
spam = ['cat', 'bat', 'rat', 'elephant']
print(str('cat' in spam))
print(str('dog' in spam))
print(str('dog' not in spam))
```

True  
False  
True

# Precedence and associativity

Operator precedence and associativity decide the priorities of the operator.

- **Operator Precedence:** This is used in an expression with more than one operator with different precedence to figure out which operation to perform first.
  
- **Operator Associativity:** If an expression has two or more operators with the same precedence, then Operator Associativity is used to find. It can either be Left to Right or from Right to Left.

Cont..

## Precedence and Associativity Table –

Operator	Description	Associativity
( )	Parentheses	Left to Right
**	Exponent	Right to Left
* / %	Multiplication, Division, Modulus	Left to Right
+ -	Addition, Subtraction	Left to Right
<< >>	Bitwise shifts	Left to Right
< <= > >= == !=	Relational operators	Left to Right

# Python 2 vs Python 3

- Python has started its journey in 1989-1990 when people started implementation on it.
- In year 2000, python 2.0 came with new features and have a healthy support to python.
- Memory management was the major part evolved in python 2.0.
- But in 2008, python has changed in a revolutionary manner to python 3.0.
- There was no support of backward compatibility in python 3.0.

# Python 2 vs Python 3

Let us have a look to the differences between Python 2 and Python 3.

- In Python 2, **print** is a **statement** syntax, but in python 3, **print** is a **built-in function**.
- In python 2, the input was taken from the user by using **raw\_input()** function.  
In python 3, **input () function** is used to take input instead of raw\_input().
- When we divide two numbers in python 2, the output is the nearest whole number. Like  $7/2$  is 3. In python 3, the fractional numeric value will be shown as  $7/2$  is 3.5.

# Python 2 vs Python 3

Let us have a look to the differences between Python 2 and Python 3.

- In for loop, the iterations are used using a **xrange function** in python 2, which is replaced by **range function** in python3.
- Some of the libraries which are available in python 2 are not moved in python 3.
- Similarly, now the developers are making new libraries for python 3 which are incompatible to python 2.

# Summary

- ❑ Python is an open-source, high level, interpreter- based language that can be used for scientific and non-scientific computing purposes.
- ❑ Comments are non-executable statements in a program.
- ❑ An identifier is a user defined name given to a variable or a constant in a program.
- ❑ Datatype conversion can happen either explicitly or implicitly.
- ❑ Operators are constructs that manipulate the value of operands.
- ❑ Python has input() function for taking user input.
- ❑ Python has print() function to output data to a standard output device.
- ❑ There are several data types in Python — integer, boolean, float, complex, string, list, tuple, sets, None and dictionary.

# References

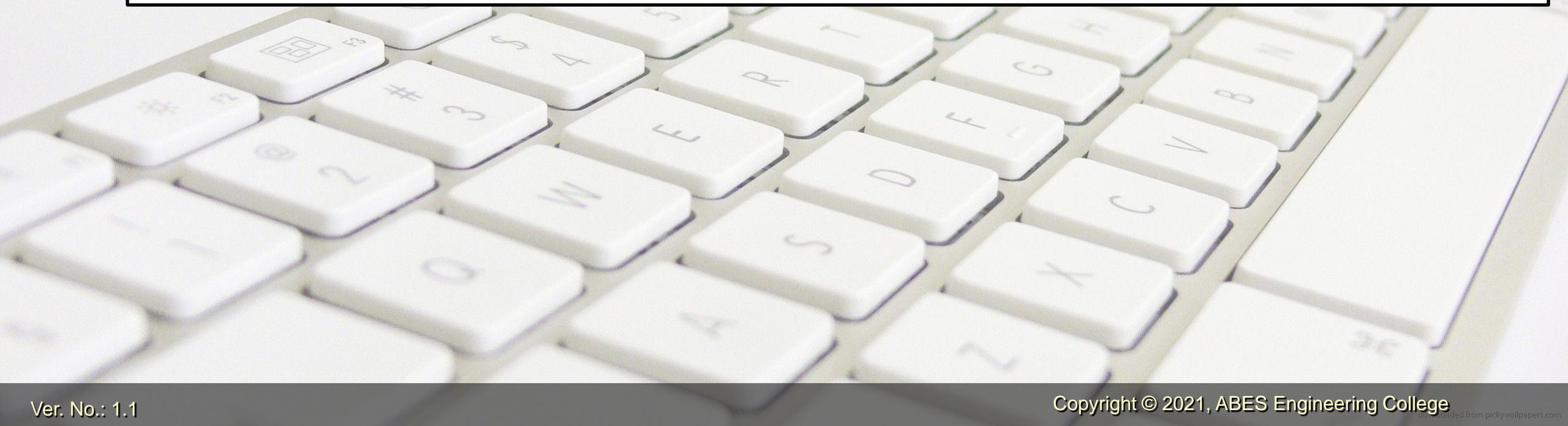
1. <https://docs.python.org/3/tutorial/controlflow.html>
2. Think Python: An Introduction to Software Design, Book by Allen B. Downey
3. Head First Python, 2nd Edition, by Paul Barry
4. Python Basics: A Practical Introduction to Python, by David Amos, Dan Bader, Joanna Jablonski, Fletcher Heisler
5. <https://www.fullstackpython.com/turbogears.html>
6. <https://www.cubicweb.org>
7. <https://pypi.org/project/Pylons/>
8. <https://www.upgrad.com/blog/python-applications-in-real-world/>
9. <https://www.codementor.io/@edwardbailey/coding-vs-programming-what-s-the-difference-yr0aeug9o>

# Thank You

---

# Control Flow System

## Day 1



# General Guideline

© (2021) ABES Engineering College.

This document contains valuable confidential and proprietary information of ABESEC. Such confidential and proprietary information includes, amongst others, proprietary intellectual property which can be legally protected and commercialized. Such information is furnished herein for training purposes only. Except with the express prior written permission of ABESEC, this document and the information contained herein may not be published, disclosed, or used for any other purpose.

# Objective of Control Flow System

**To understand the purpose and syntax of decision control statements.**

**To understand the purpose and syntax of Iterative/looping statements and jump statements.**

**To develop python programs with conditions and loops.**

**To apply the knowledge of conditions, loops and jump in solving real time problems.**

**To create python programs with decision ,looping and jump statements.**

# Topics Covered

Day 1

- 2.1 Introduction**
- 2.2 Selection / Decision control statement**
  - Simple If statement
  - If Else statement
  - If-Elif statement
  - Nested If Else statement

Day 2

- 2.3 Iterative/ Looping Statements**
  - While loop
  - Nested While loop

Day 3

- 2.3 Iterative/ Looping Statements**
  - Range() Function
  - For loop
  - Nested for loop

Day 4

- 2.4 Jump Statements**
  - Break
  - Continue
- 2.5 Else with Loop**
  - While with else
  - For with else

# Session Plan - Day 1

## 2.1. Introduction to Decision Control Statements

## 2.2. Selection / Decision control statement

- Simple If Statements
- If Else Statements
- If ..elif..else statements
- Nested If Else Statements
- Examples
- Review questions
- Summary

# Introduction

Control flow of the program is:

The order in which the **code of the program** executes.

Regulated by:

- Conditional statements.
- Iterative/looping statements.

## Contd..

□ **Conditional statement examples:** Based on certain conditions, we take **decisions** in our daily life.

- Darkness in the room: Turn on lights.
- No Darkness: Do Nothing
- Weather is *Cold*: **Drink Coffee**
- Weather is *not Cold*: **Softdrink**



## Contd..

- ❑ *Iterative Looping statements examples:* For taking **decisions** we perform various actions and **repeat** the same **action** many **times**.

*Real Life Scenario:*

- ❑ Suppose you want to buy a new t-shirt, visit the shop.
- ❑ If you do not found a shirt of your choice, visit to the new shop.
- ❑ Until your desired shirt is found, action is performed again and again.
- ❑ This is called looping.



# Selection/decision control statements

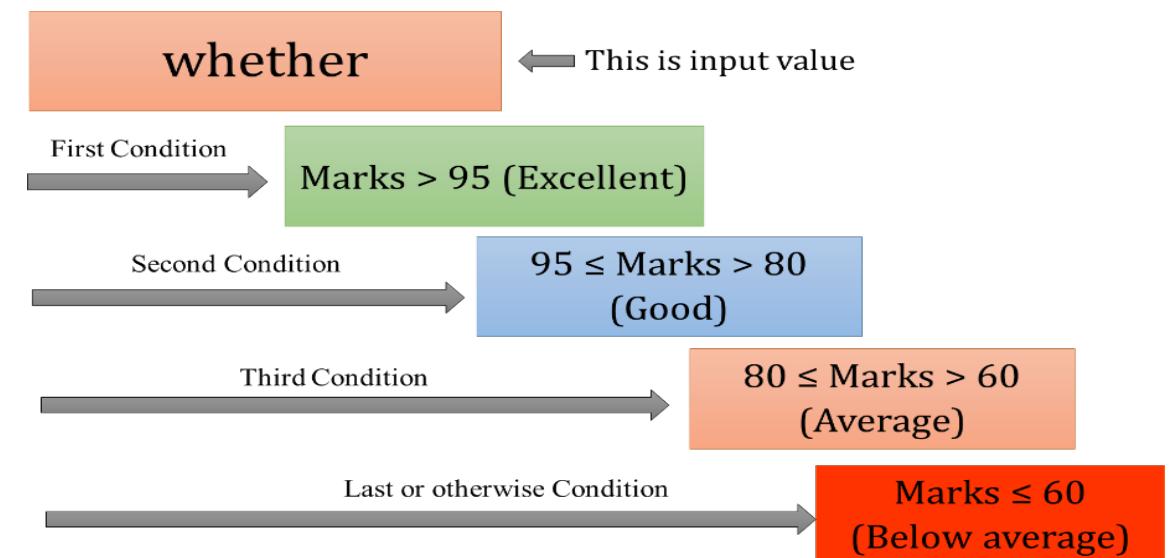
- ❑ It is used to control the flow of execution of program depending upon condition:

- if the condition is **true** then the code block **will execute**
- if the condition is **false** then code block will **not execute**.

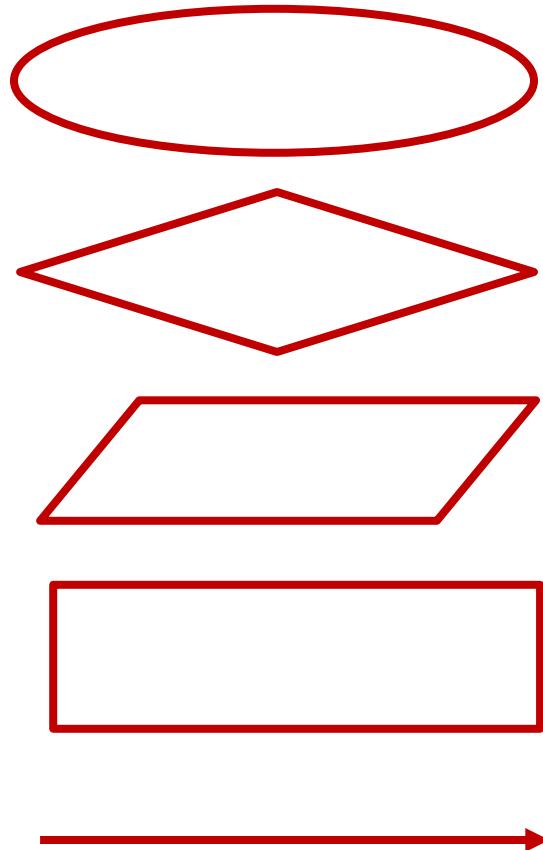
*Real Life Scenario:*

Consider the grading system for students:

- Excellent : Above 95%.
- Good: Greater than 80% & less than equal to 95%.
- Average: Greater than 60% & less than equal to 80%.
- Below average: Less than 60%



# Revision of Flowchart



Used for start and stop.

Used for decision making.

Used for input/output.

Used for processing statement.

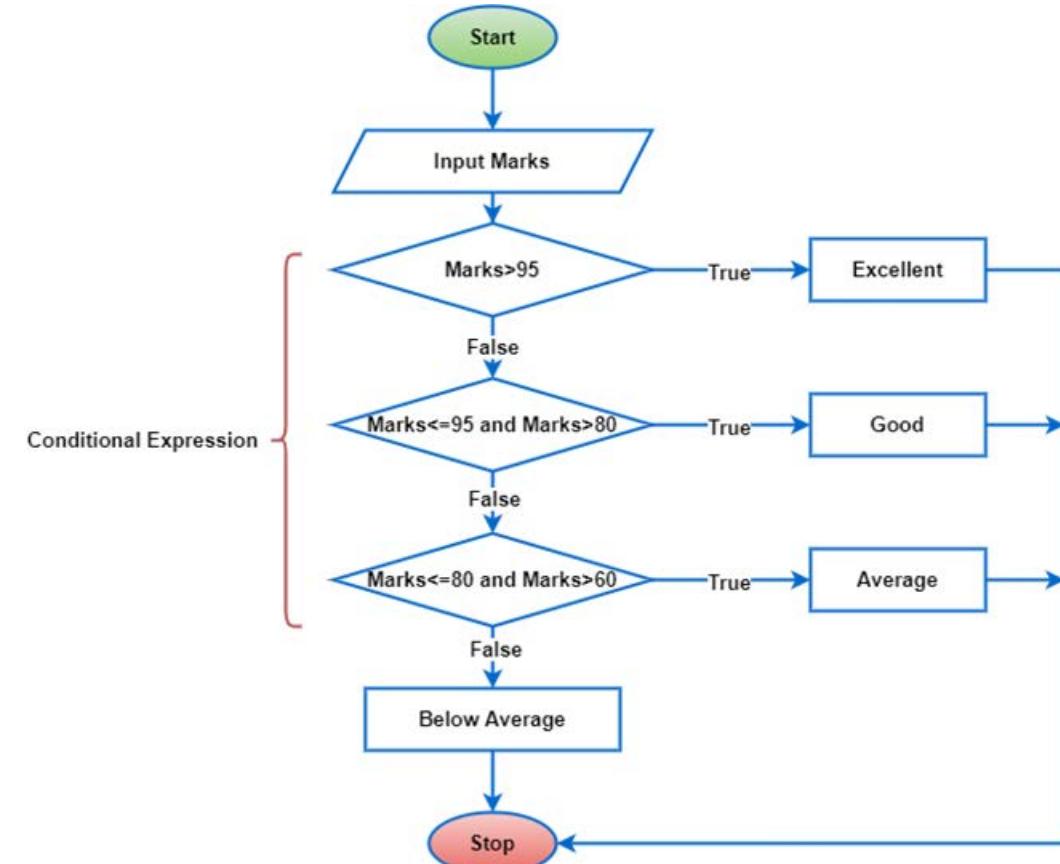
Connector / flow

# Flowchart of the grading system

On the basis of the **conditions**, corresponding **block** of the code will be **executed** or not.

Consider the **grading system** for students:

- Excellent :Above 95%.
- Good :Greater than 80% & less than equal to 95%.
- Average : Greater than 60% & less than equal to 80%.
- Below average: Less than 60%

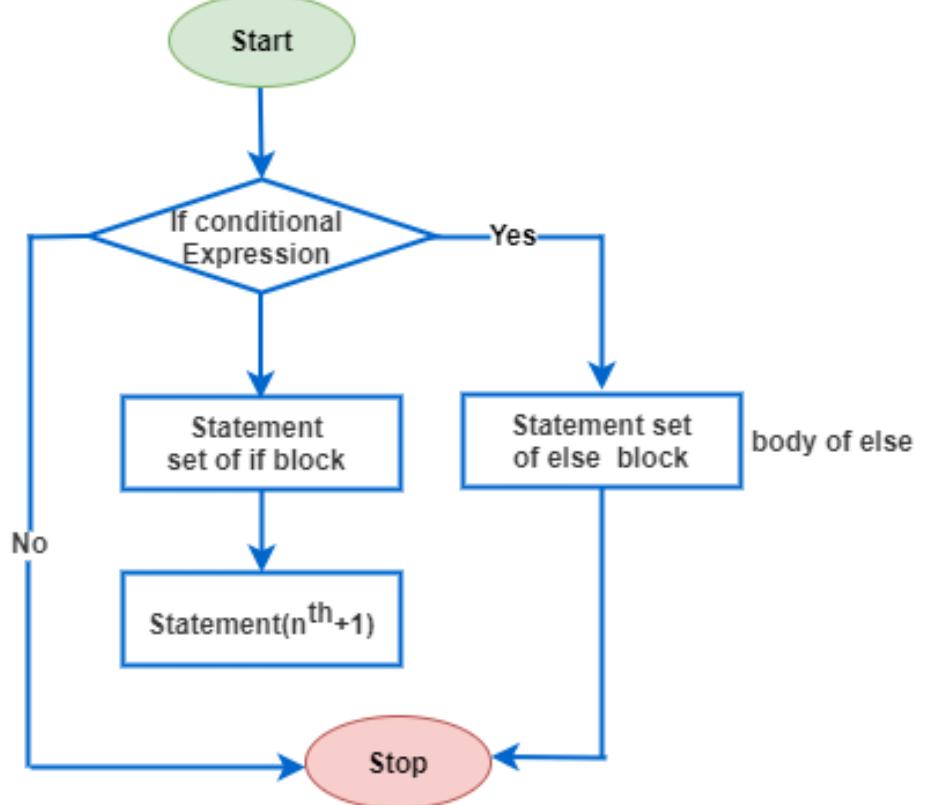


# Types of decision/selection control statements:

- Simple if
- if-else
- if..elif..else
- Nested...if...else

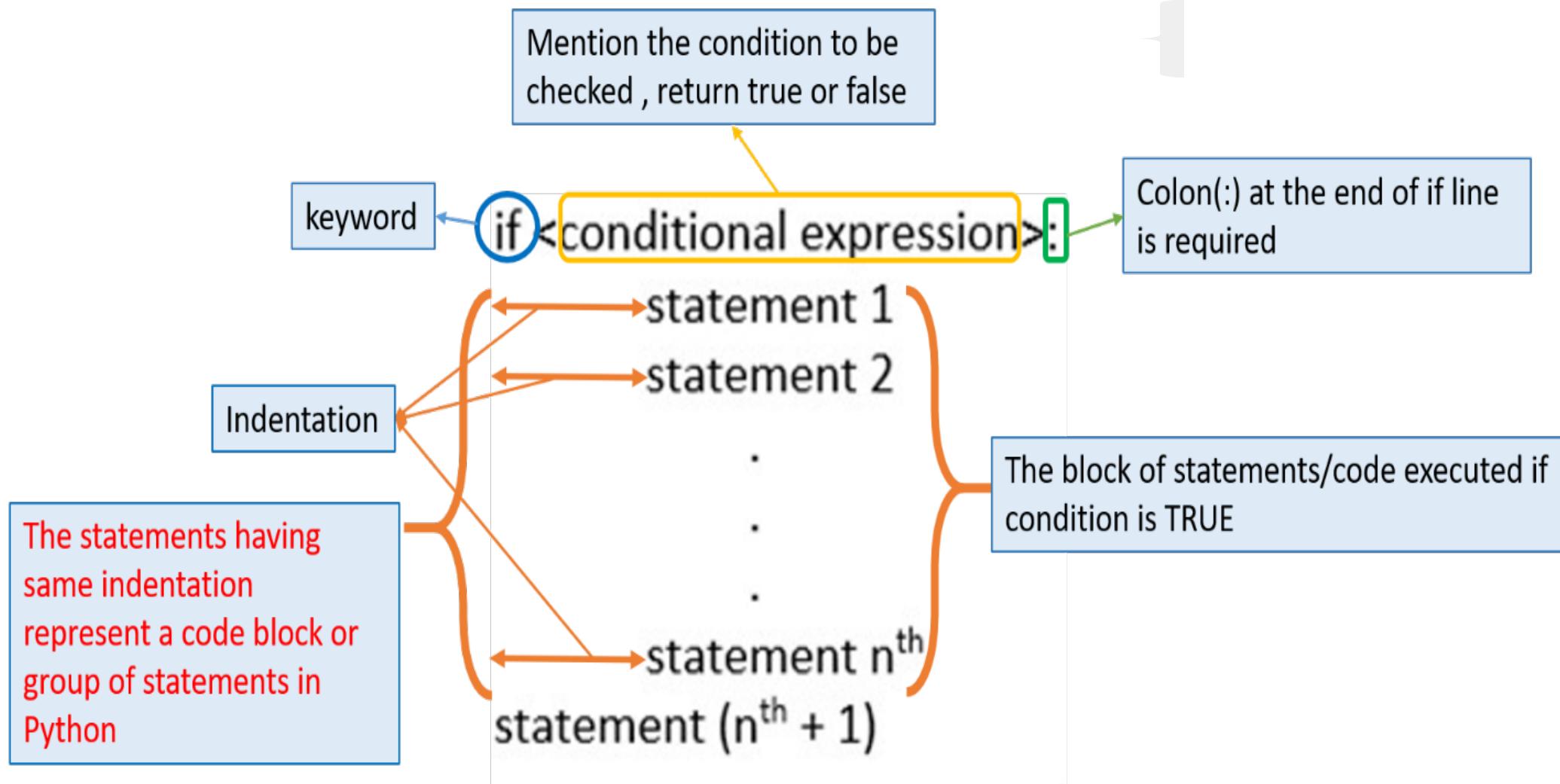
# Simple if statement

- *if* evaluates whether a condition is:  
True  
False.
- *if* block is executed:  
if and only if the value of its condition expression is true.



**Note:** As we use curly braces in 'C' Language to define the scope of conditions and loops; in the same manner we use colon (:) and indentation in Python.

# Syntax of Simple if



# Examples

- Print whether a number is even or not.

Case-1 : When n is even.

```
n=10 #Initialize the value of n

if(n % 2 == 0):#test condition for n is even
    print("n is an even number") ← Execute only if condition
                                evaluates to True
    print("Statement outside if block") ← It always run either if
                                         True or False

n is an even number
Statement outside if block } Output
```

Case-2 : When n is not even

```
n=11 #Initialize the value of n

if(n % 2 == 0):#test condition for n is even
    print("n is an even number") ← Execute only if condition
                                evaluates to True
    print("Statement outside if block") ← It always run either if
                                         True or False

Statement outside if block } Output
```

# Examples

- A program to **increment** a number if it is **positive**.

```
x = 10 #Initialize the value of x
if(x > 0): #test the value of x
    x = x+1 # Increment the value of x if it is > 0
    print(x)
print("Statement outside if block") ← It always run either if True or False
```

} Execute only if condition evaluates to True

11  
Statement outside if block } Output

## Explanation:

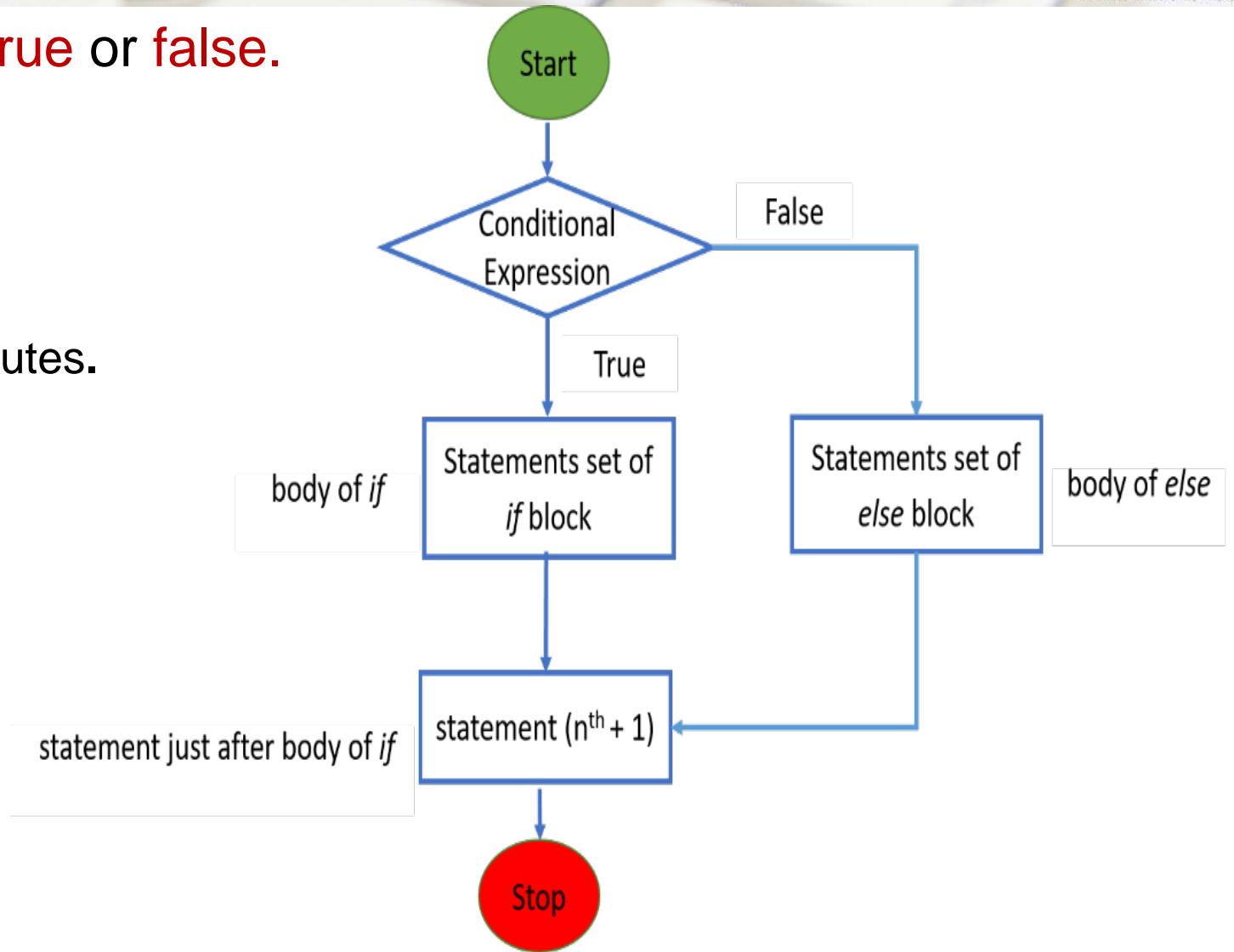
- The condition  **$x>0$**  is **True** for value of  **$x = 10$**  so the statements in the block of code will be executed.
- The statement  **$x = x+1$**  will increment the value of  **$x$**  by 1 i.e.  **$x = 10 + 1$** ;  **$x = 11$**  and statement **`print(x)`** will print  **$x$**  value as **11**.

# if else Statement

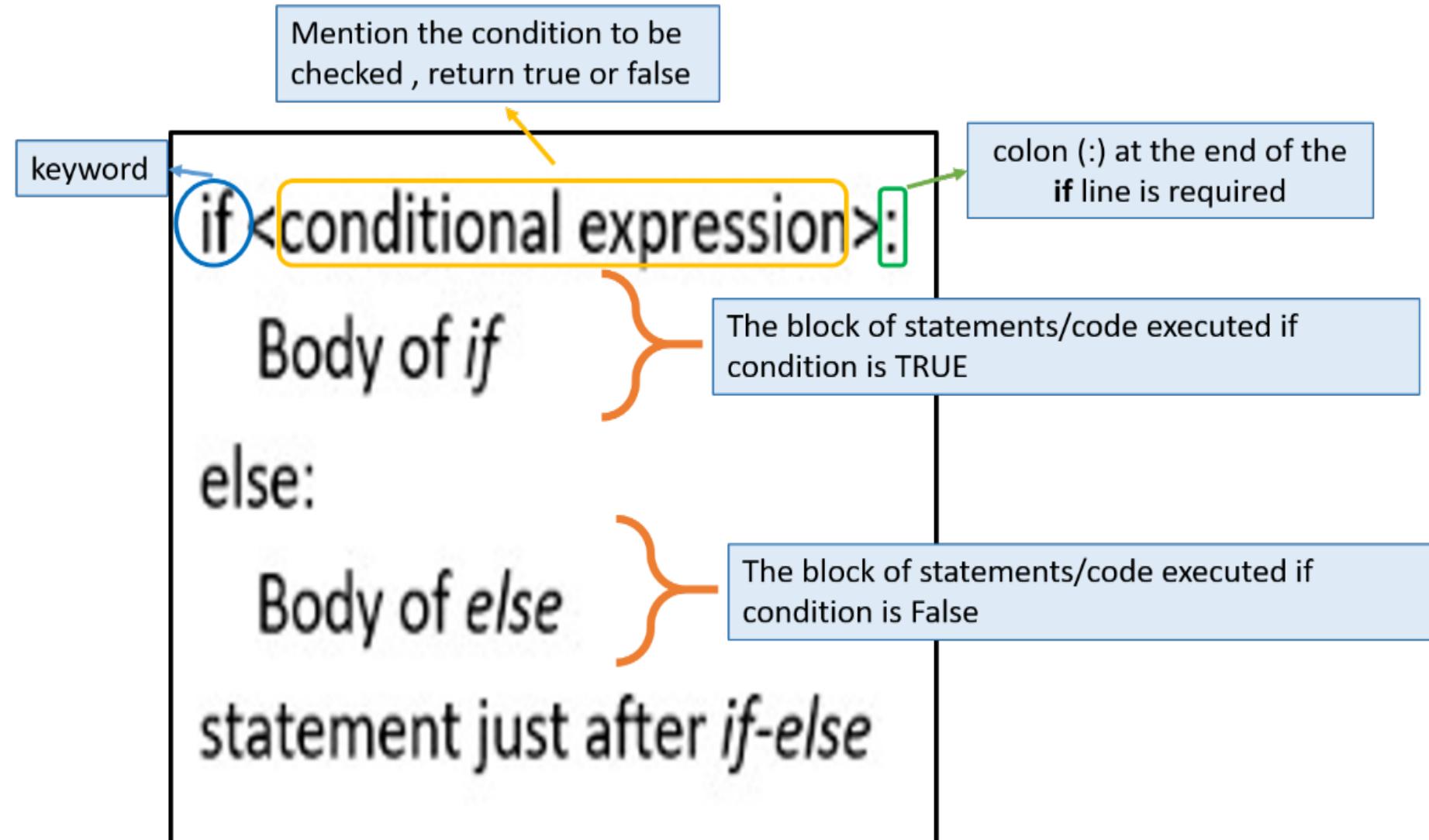
□ It checks whether a condition is **true** or **false**.

□ If a **condition** is **true**,

- The ***if*** statement executes
- Otherwise, the ***else*** statement executes.

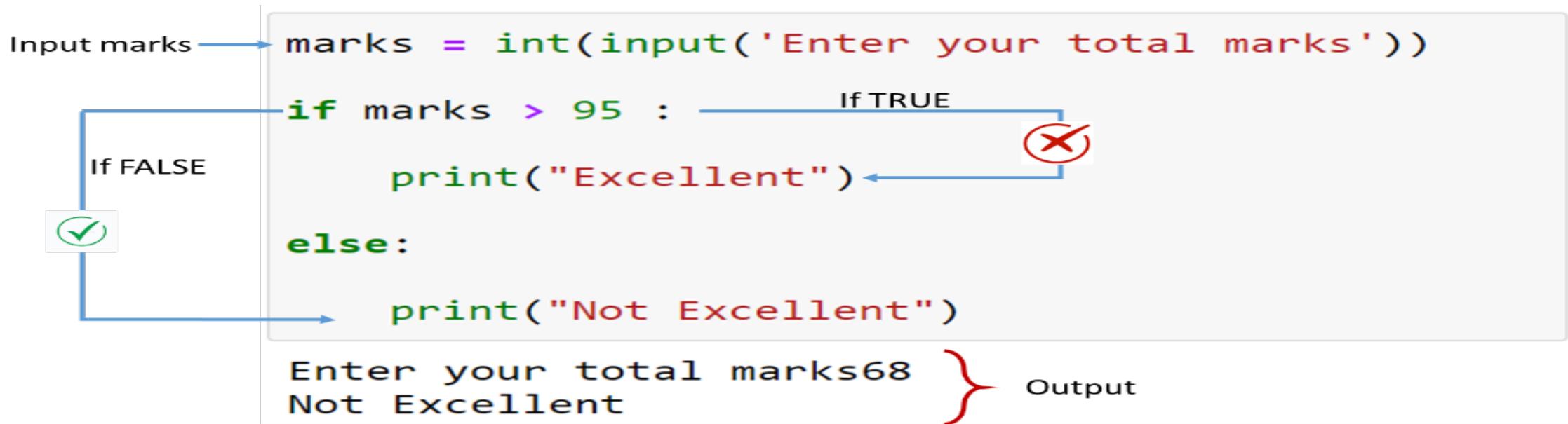


# Syntax of if else Statement



# Example

- A program to print “Excellent” if marks is greater than 95 otherwise print “Not Excellent”



Explanation:

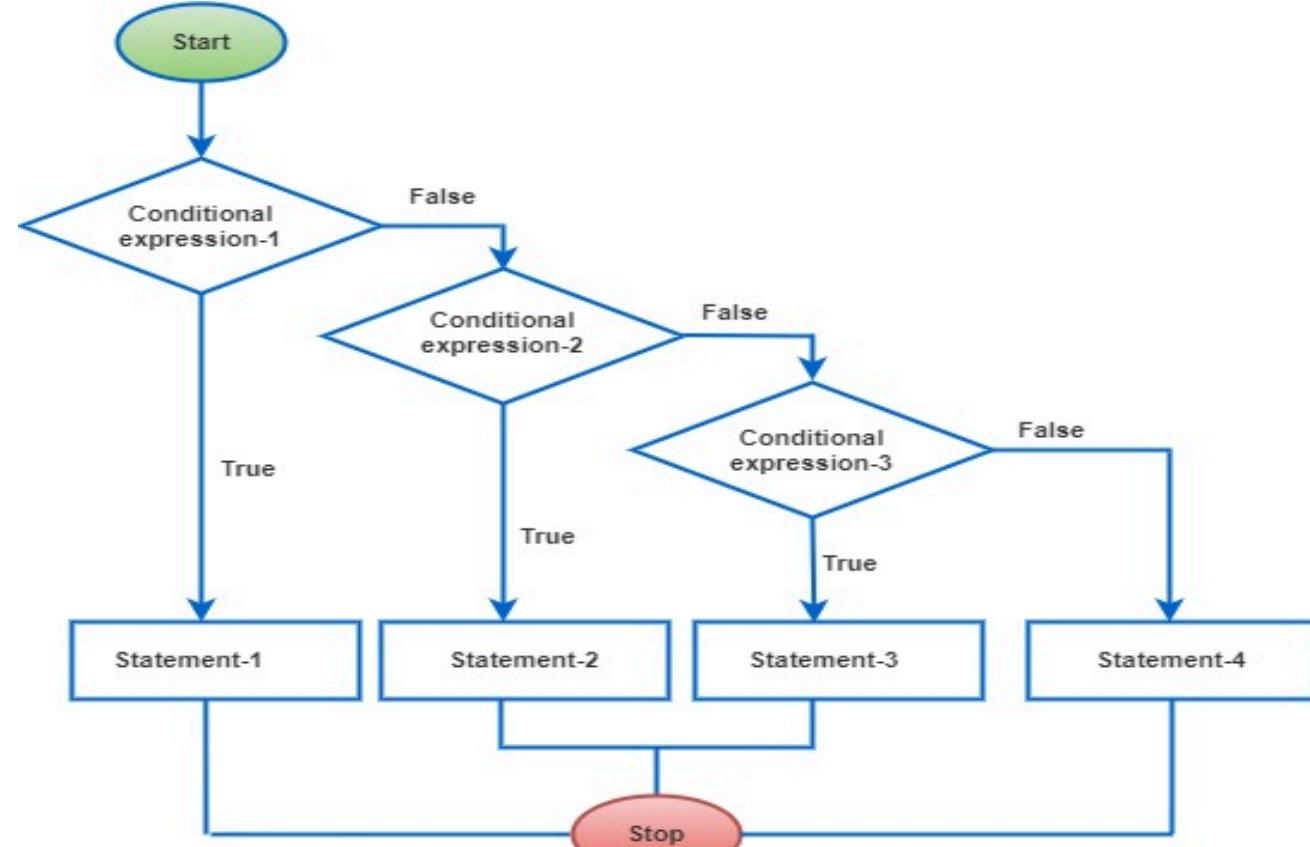
- For input 68, condition `marks > 95` is False. Therefore, else statement `print ("Not Excellent")` is executed and output will be “Not Excellent”.

# if..elif..else statement

Extension of the **if-else** statement

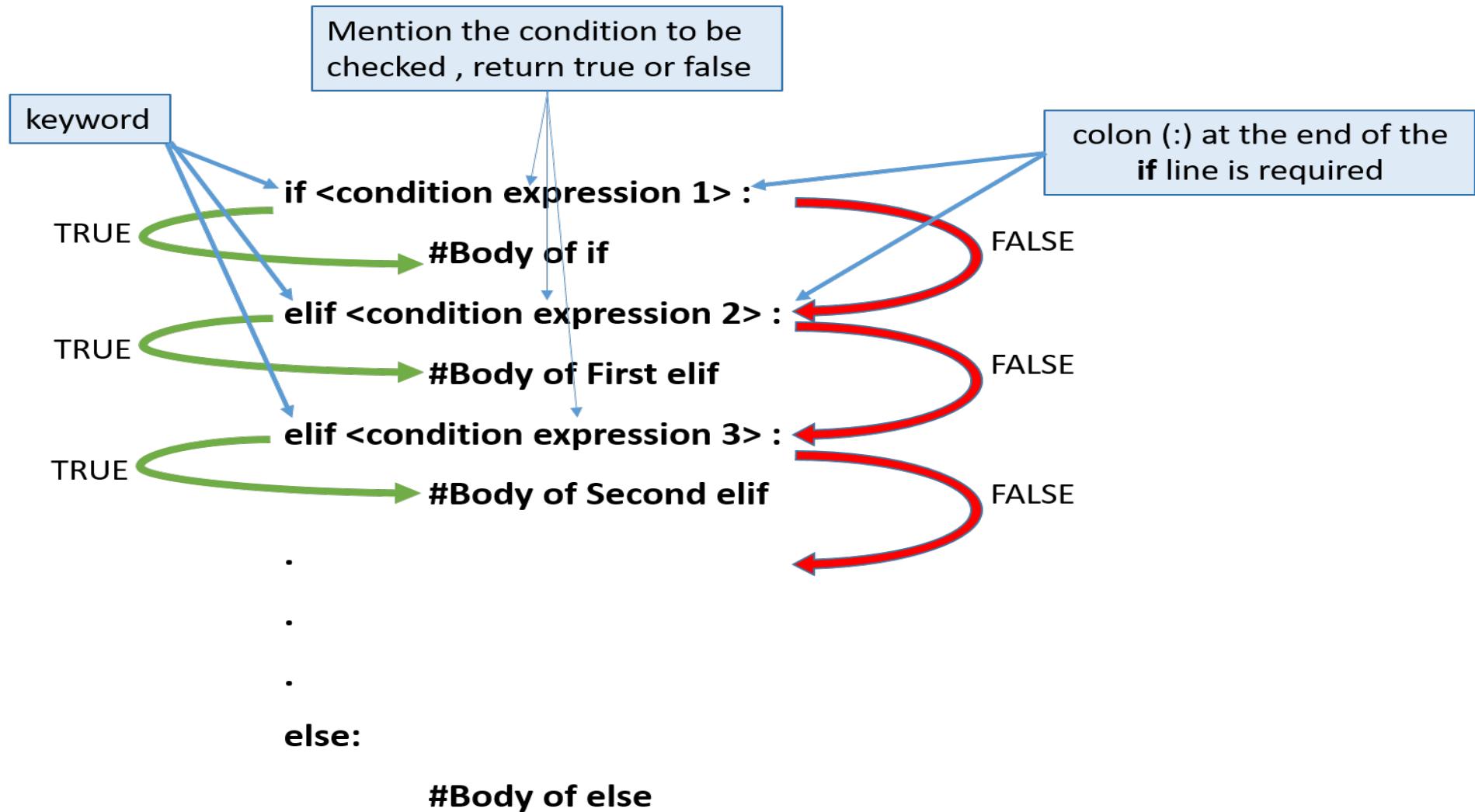
For a multiple conditional statements

- We use **if-elif** statement.



Note: The conditional expression of **if-elif-else** are executed from top to bottom in a sequential manner. An **elif** statements are known as [elif Ladder](#).

# Syntax of if..elif..else statement



# Example

```

marks = int(input('Enter your Percentage'))
if marks > 95 :           ← For marks = 88 this will evaluate FALSE
    print("Excellent")
elif marks > 80 and marks <= 95: ← For marks = 88 this will
    print("Good")           evaluate TRUE
elif marks > 60 and marks <= 80: ← Once conditional statement
    print("Average")        evaluated TRUE, then
else:                      All conditional statement
    print("Below Average")  after that will not be executed
  
```

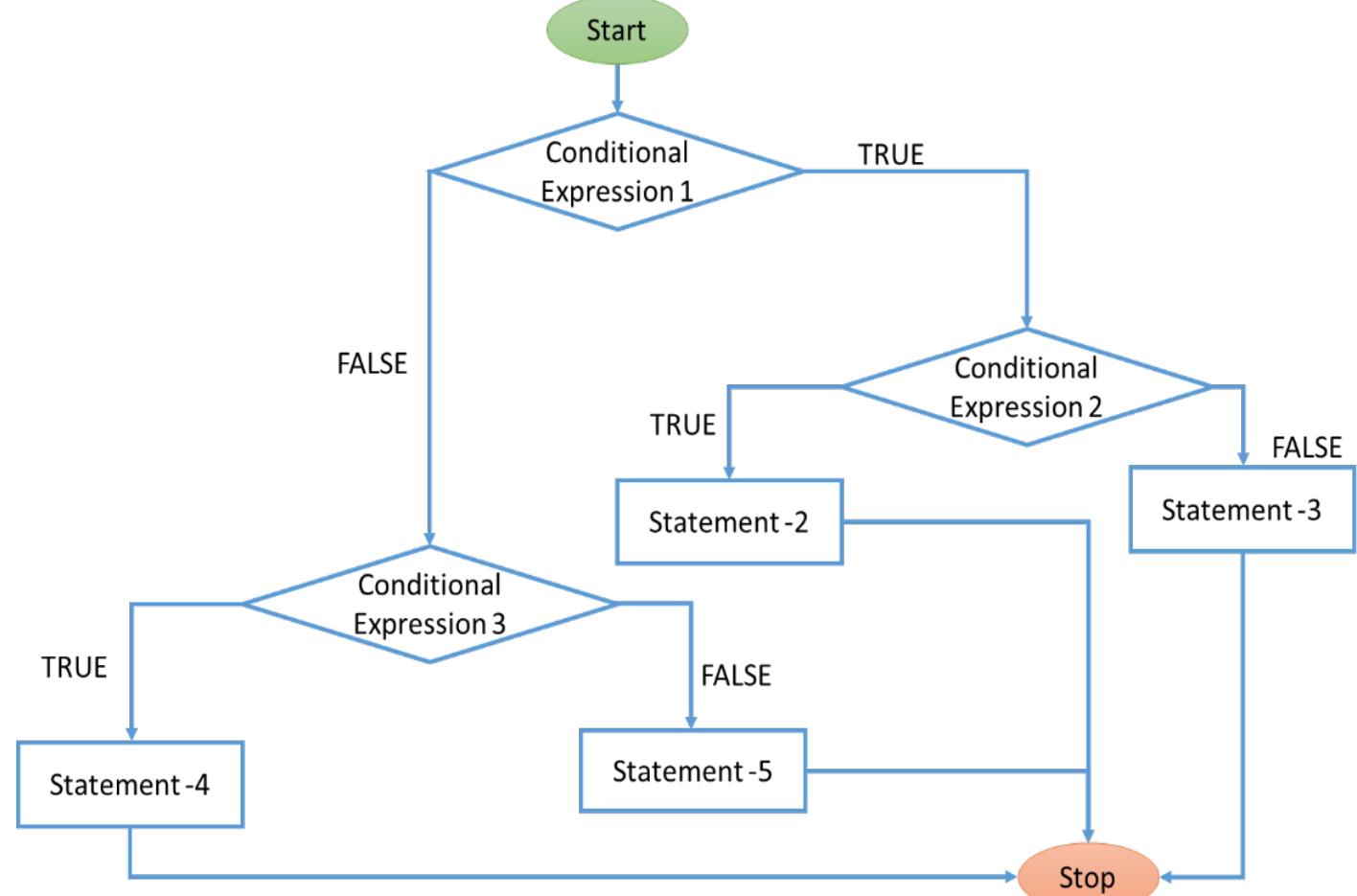
Enter your Percentage88  
Good

## Explanation:

- ❑ For input 88, condition marks > 95 is False & control goes to next elif statement (marks>80 and marks<=95), which is True and statement print ("Good") will be executed.
- ❑ The output will be “Good”.

# Nested if-else statement

- When an **if-else statement** is present inside the body of another “if” or “else” then this is called nested if else.



# Syntax of Nested if..else statement

```
if <conditional expression 1> :  
    if <conditional expression 2> :  
        #Statement 2  
    else:  
        #Statement 3
```

↳ Executes when condition1 is TRUE

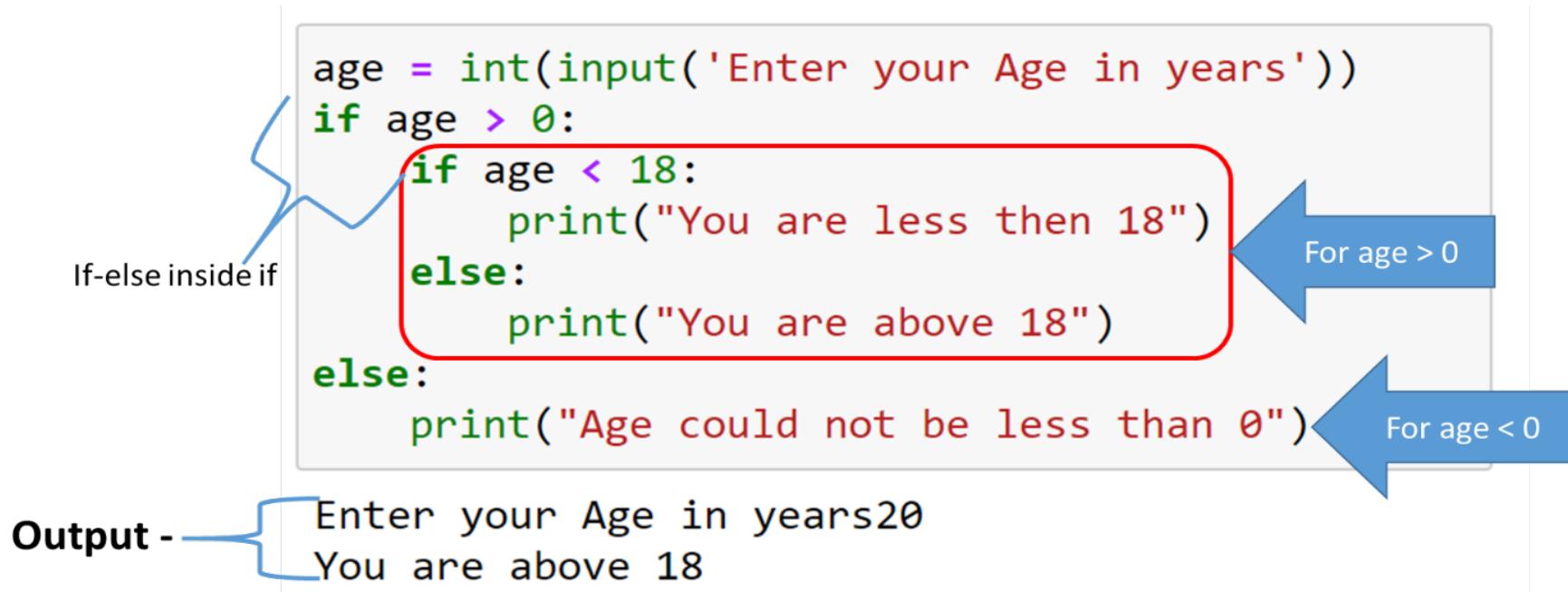
```
else :  
    if <conditional expression 3> :  
        #Statement 4
```

↳ Executes when condition1 is FALSE

```
    else:  
        #Statement 5
```

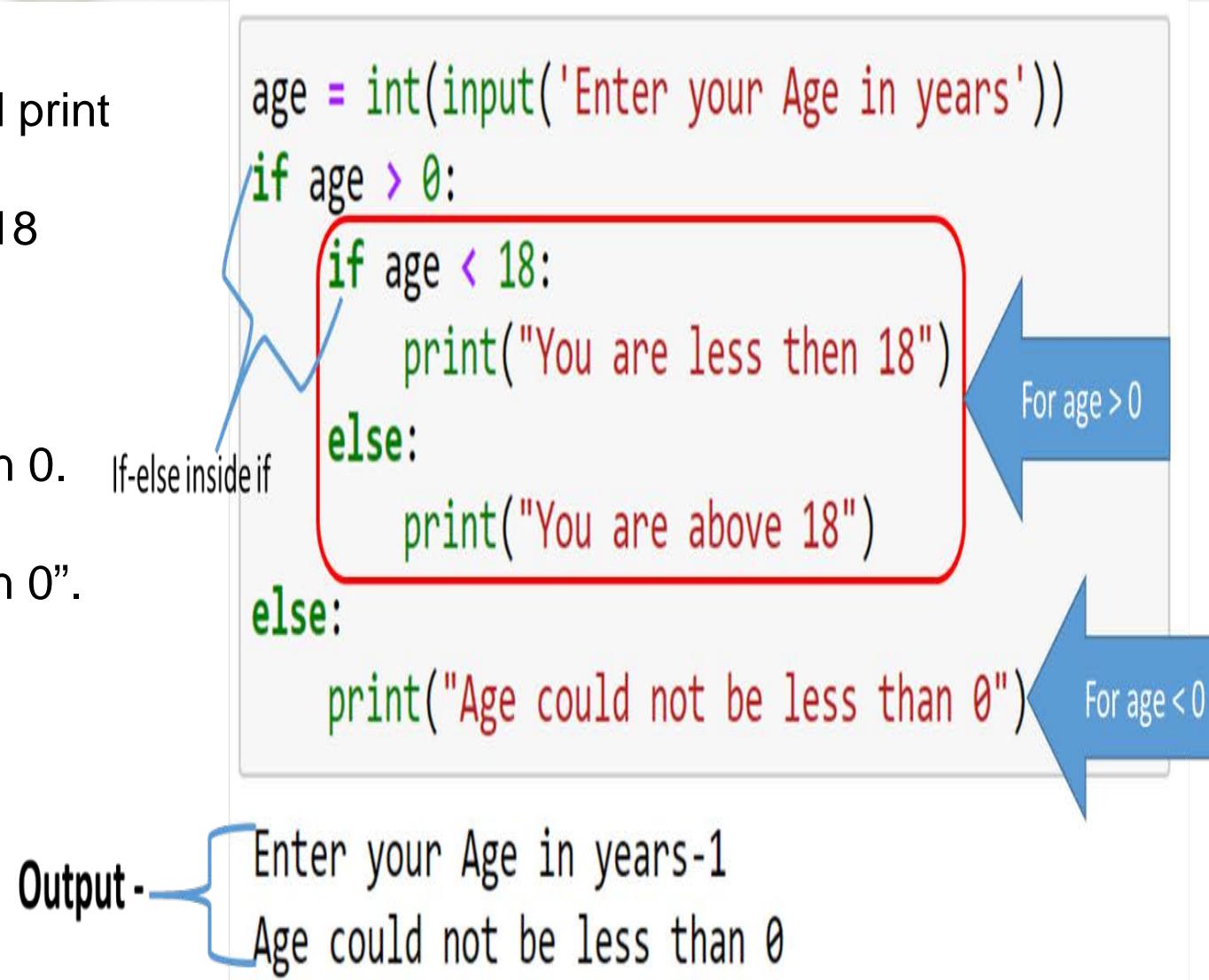
# Example

- A program to take age as input from user and print “You are less than 18” if age is less than 18, otherwise print “You are above 18”. Also check that age could not be less than 0. If so then print “Age could not be less than 0”.



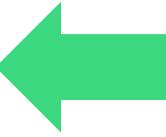
# Example

- A program to take age as input from user and print
  - “You are less than 18” if age is less than 18
  - otherwise print “You are above 18”.
  - Also check that age could not be less than 0.
  - If so then print “Age could not be less than 0”.



# Can you answer these questions?

1. Which one of the following is a valid Python if statement :

- A) **if** a>=2 : 
- B) **if** (a >= 2)
- C) **if** (a => 22)
- D) **if** a >= 22

3.Which of following is not a decision-making statement?

A) **if-elif** statement

B) **for** statement

C) **if -else** statement

D) **if** statement



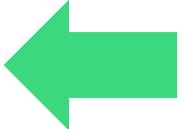
#### 4. Predict the output of the following code:

- A) No output
- B) okok
- C) ok
- D) None of above

```
x=3
if (x>2 or x<5 )and x==5:
    print("ok")
else:
    print ("no output")
```

5. What is the output of the following code snippet?

- A) Launch a Missile
- B) Let's have peace
- C) 0.3
- D) None



```
y=0.3
if y!=0.3:
    print("lunch a missile")
else:
    print ("let's have peace")
```

6. Which of the following is true about the code below?

```
x = 3
if (x > 2):
    x = x * 2;
if (x > 4):
    x = 0;
print(x)
```

- A) x will always equal 0 after this code executes for any value of x
- B) if x is greater than 2, the value in x will be doubled after this code executes
- C) if x is greater than 2, x will equal 0 after this code executes



# Summary

- Control statement are statements that control the flow of execution of statements so that they can be executed repeatedly and randomly.
- The if statement executes a group of statements depending upon whether a condition is true or false.
- The if..else statement executes a group of statements when a condition is true; Otherwise, it will execute another group of statements.
- The if..elif statement is an extension of the if-else statement. When we have multiple conditional statements, then we use if-elif statement.
- When an if..else statement is present inside the body of another “if” or “else” then this is called nested if else.

# Session Plan - Day 2

## 2.3 Iterative looping Statements

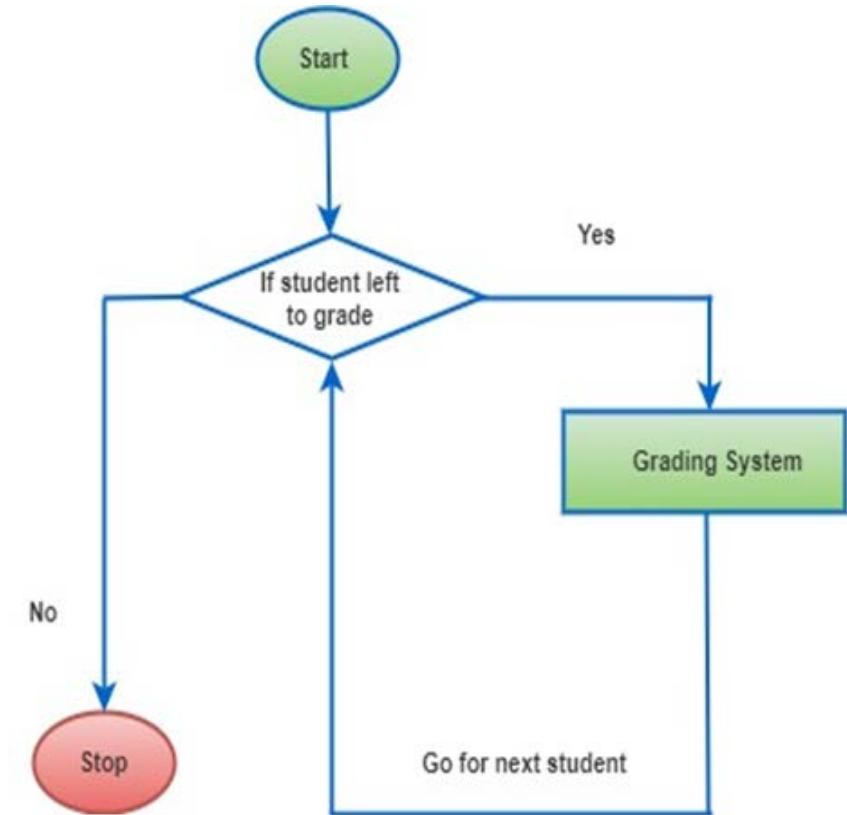
- While loop
- Nested While loop
- Examples
- Review Questions
- Summary

# Iterative/Looping Statements

Sometimes we need to perform certain operations again and again.

Real Life Scenario:

- A teacher decide to **grade 75 students** on the basis of **marks**.
- He/she wants to do this for **whole class**.
- Teacher would repeat **grading** procedure for **each student** in the class.
- This is called **iterative/ looping**.

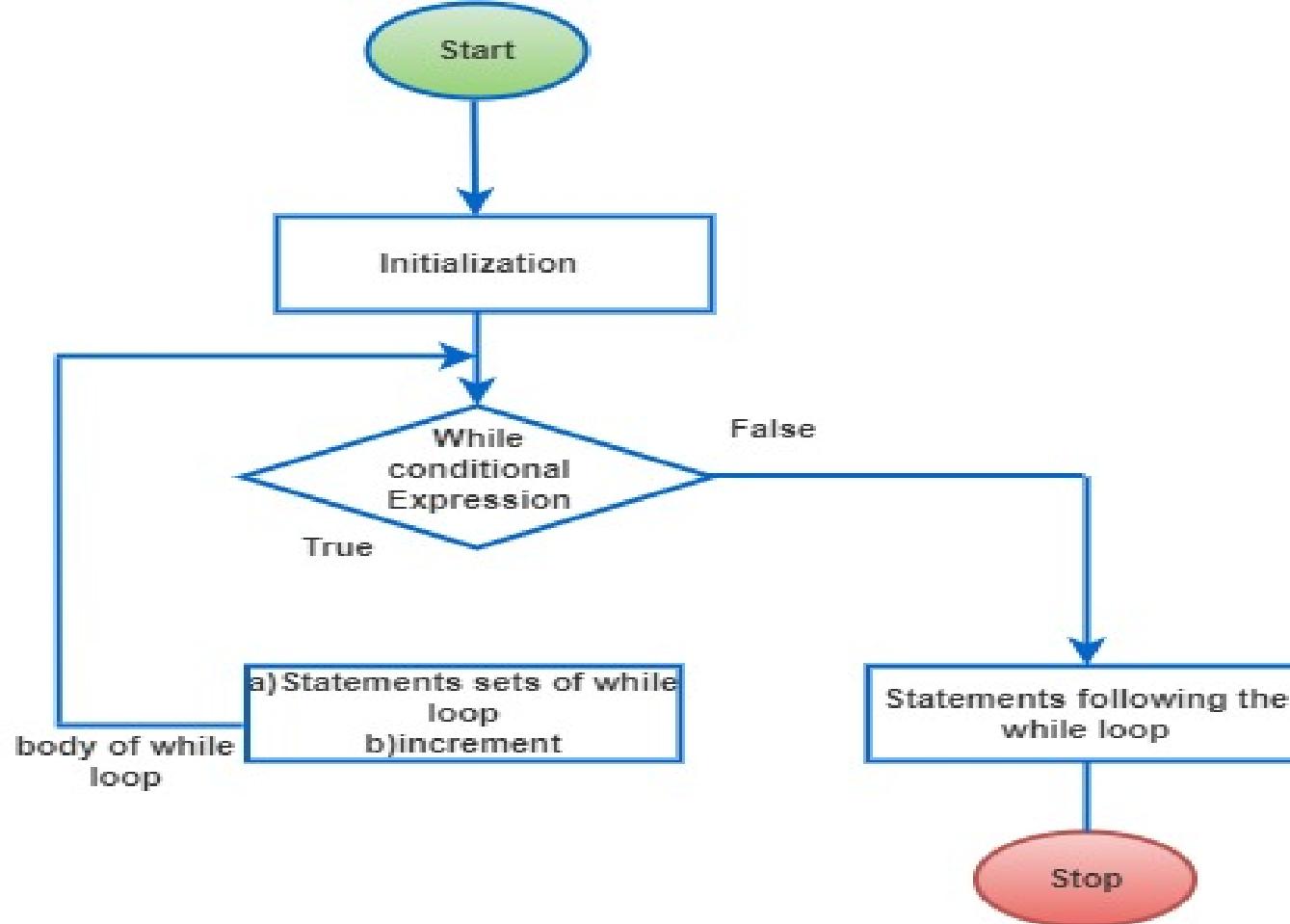


# Types of loops in Python

- While loop
- Nested While loop
- For loop
- Nested for loop

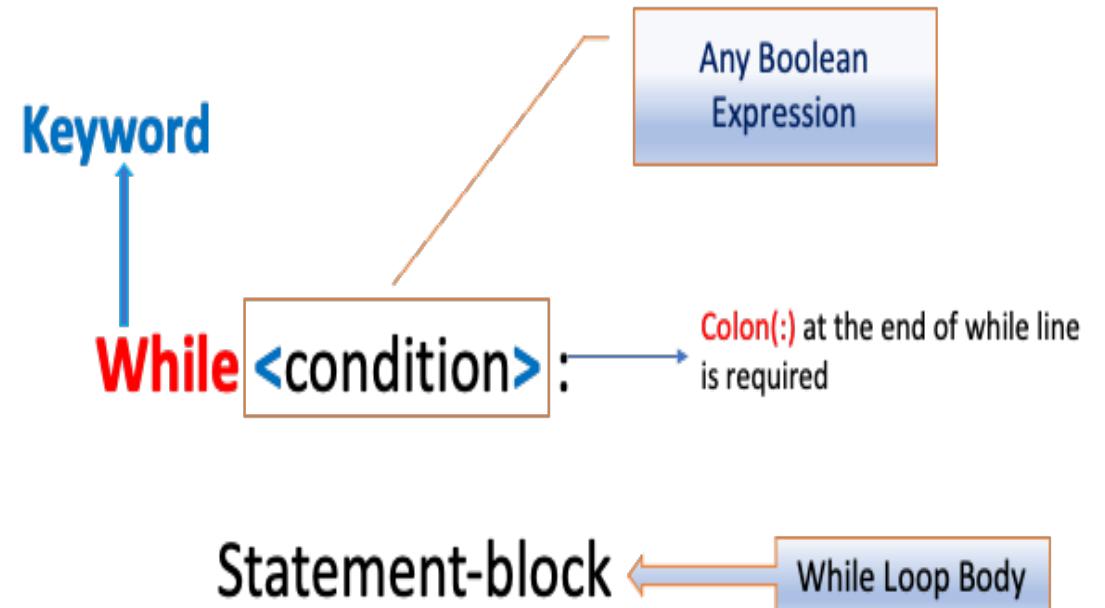
# While loop

- It is used to repeat a **block of code** as long as the given **condition** is true.



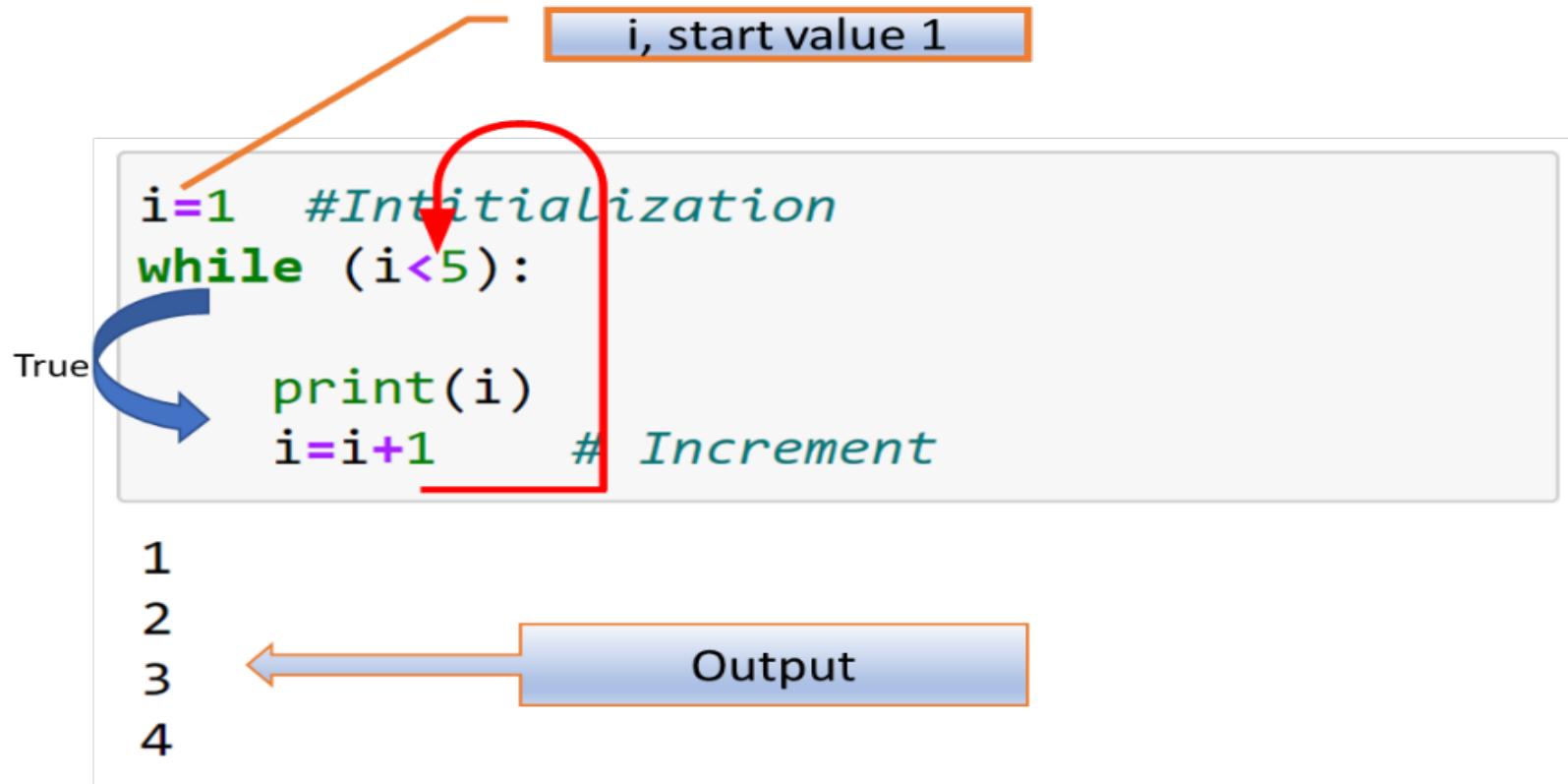
# Syntax of While loop

- ❑ Boolean expression is checked first.
- ❑ The body of the loop is entered only if the Boolean expression evaluates to True.
- ❑ After one iteration, the Boolean expression is checked again.
- ❑ This process continues until the Boolean expression evaluates to False.



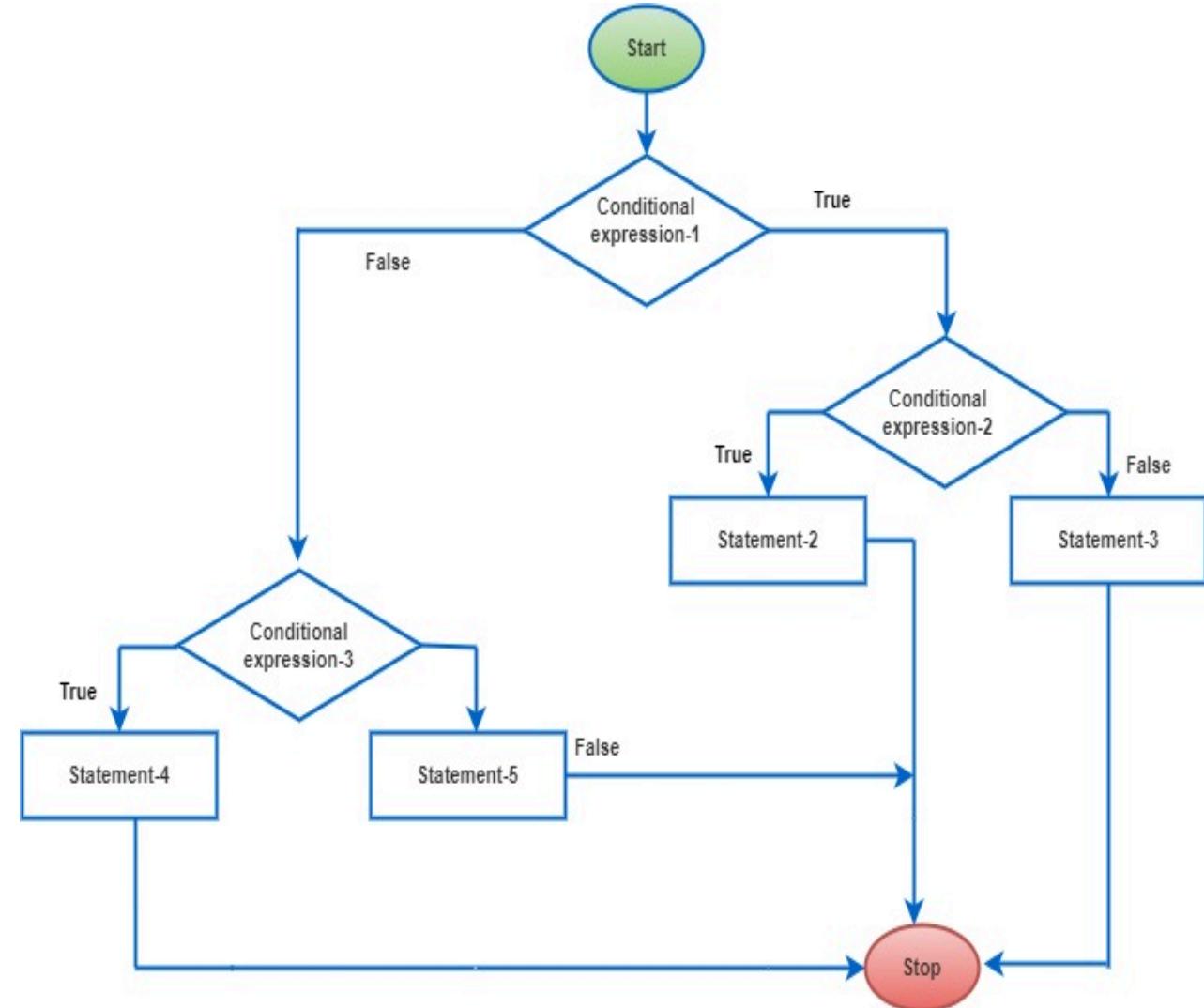
# Example

Write a Python Program to print 4 Natural Numbers using i.e. 1,2,3,4 using While loop.



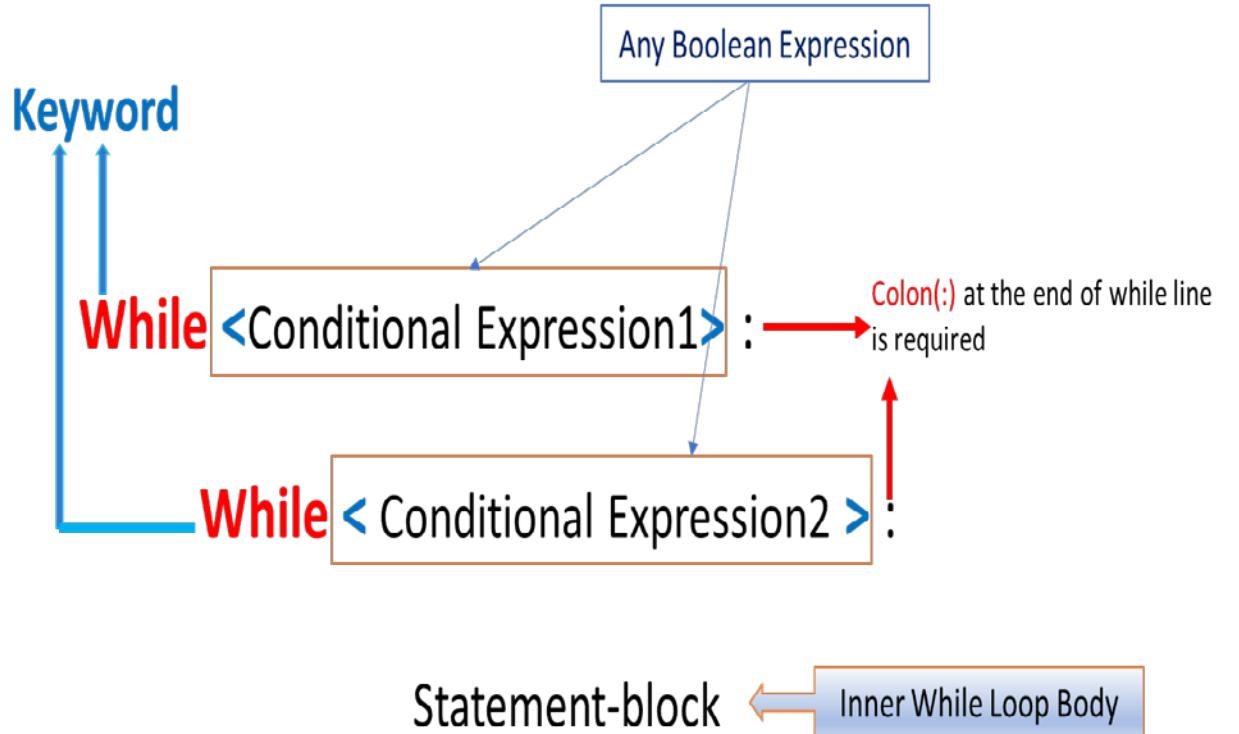
# Nested While loop

- Nested while loop is called when we use while loop inside a while loop.
- We can use any number of while loop inside a while loop.
- Main while loop as outer while loop and nested while loop as inner while loop.



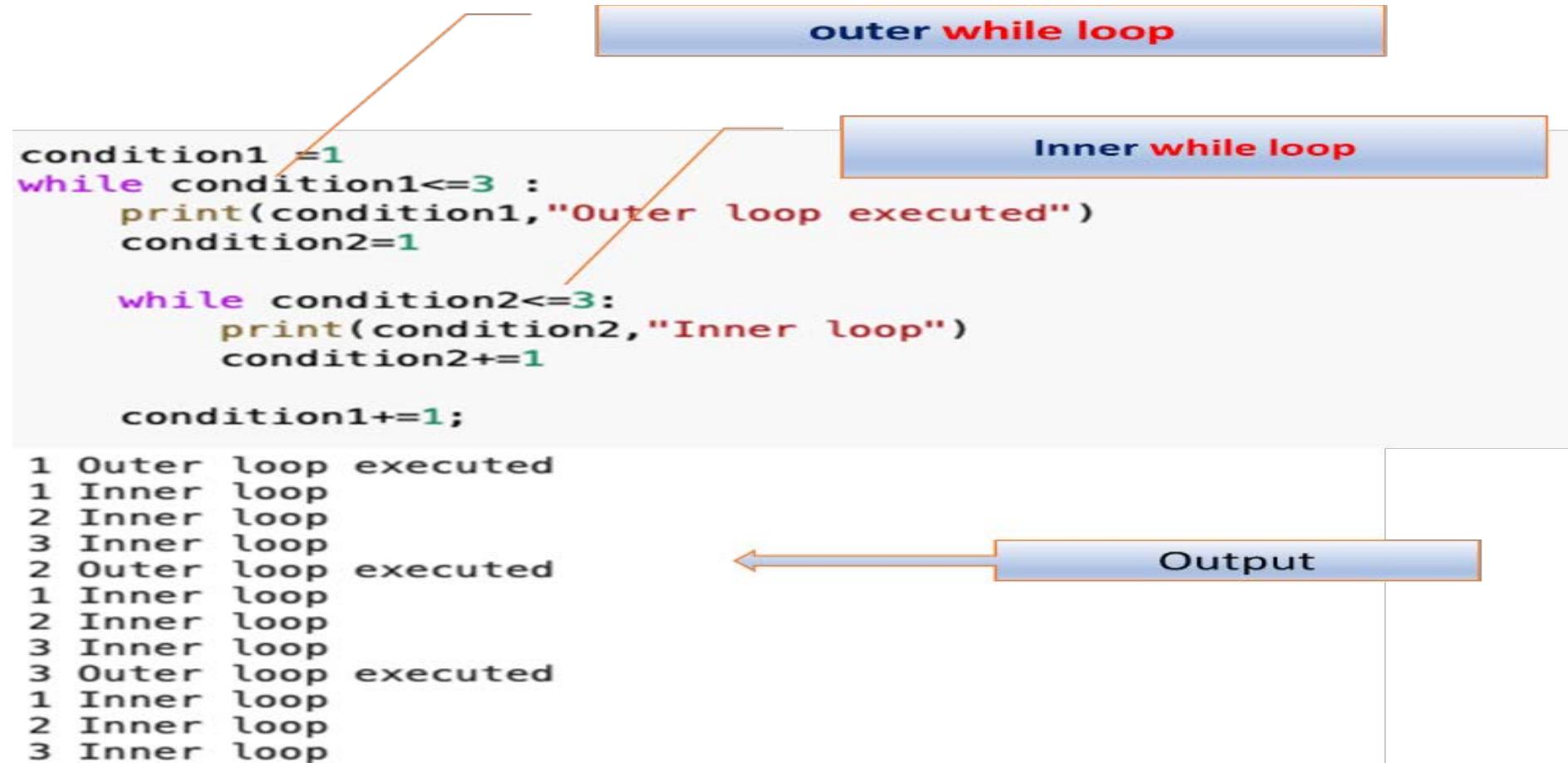
# Syntax of Nested While loop:

- Outer while loop runs  $m$  number of times.
- Inner while loop runs  $n$  number of times.
- The total no. of iterations would be  $m * n$ .



# Example of Nested While loop:

- Write a program to demonstrate While loop.



# Can you answer these Questions?

1. What will be the output of the following code snippet?

```
string1 = "Python"
i = "p"
while i in string1:
    print(i, end = " ")
```

Options

- A. None
- B. Python
- C. Ppppp
- D. PPPPP



2. What will be the **output** of the following **code snippet**?

```
i = 0
while i < 3:
    print(i)
    i += 1
else:
    print(0)
```

- A. 0 1 2
- B. 2 3
- C. 0 1 2 0 ←
- D. None of the above

3. What should be the **value** of the **variables num1** and **num2** in the **code snippet** below if the output expected is **4**?

```
num1=?  
num2=?  
while(num1>=2):  
    if(num1>num2):  
        num1=num1/2  
    else:  
        print(num1)  
        break
```

A. 16,6

B. 12,5

C. 8,2

D. 16,2



# Summary:

- While loop is used to iterate block of codes repeatedly until given condition is True
- While loop present inside another while loop it is called as nested while loop.
- The nested while loop is while statement inside another while statement.

# Session Plan - Day 3

## 2.3 Iterative looping Statements

- Range()
- For loop
- Nested for loop
- Examples
- Review Questions
- Summary

# Introduction to Iterative/looping statements:



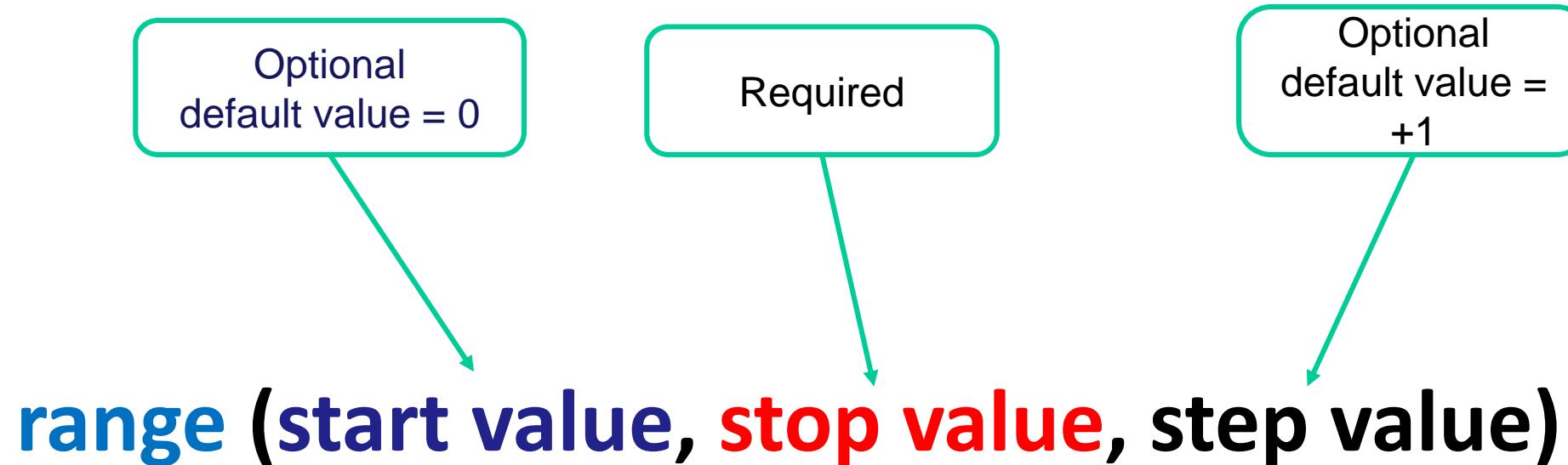
# Range() Function,

range (start value, stop value, step value)

Optional default value = 0

Required

Optional default value = +1



- Range() is a built in function in Python
- It gives the sequence of numbers.

# Range Function Example:

- ❑ range (1,5,1)
- ❑ It will generate a **sequence** starting from **value 1** up to **value 4**
- ❑ **(5 is not included)** and **step by 1**
- ❑ So, the **output** of the above gives us the sequence **1,2,3,4.**

**Note:** Range()Function returns the range object, you must typecast range function into collections.

# Range

- Range () Function with one argument.

- Single value is considered as the stop value.

- It means the start value is considered as 0, and step value is considered as 1.

Program to generate sequence numbers 0 to 4.

Typecast the range object into list



list(range(5))

Only single value

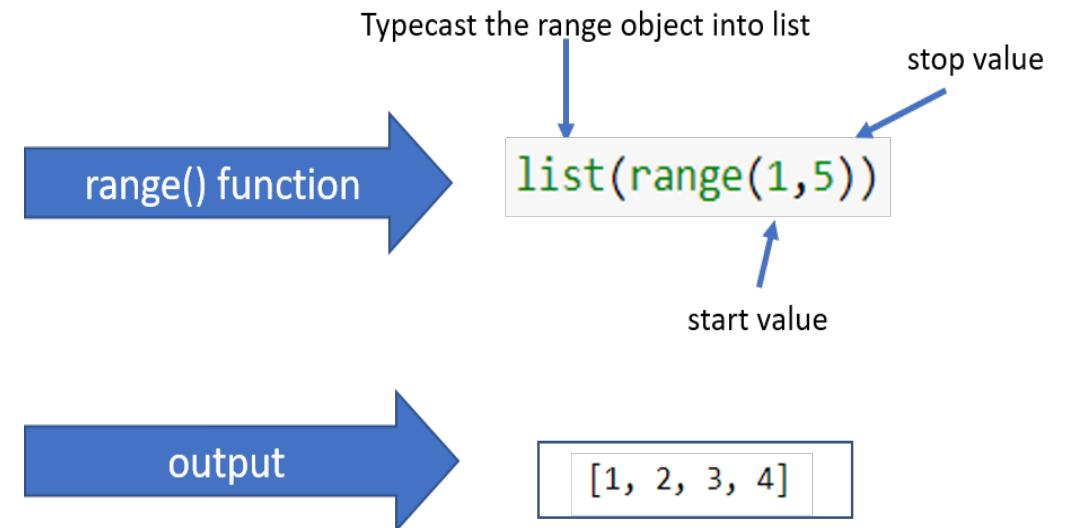


[0, 1, 2, 3, 4]

# Range

- Range () with two argument:
- This means the **start value** and **stop value** is mentioned.
- The **step values** is considered as 1 by default.

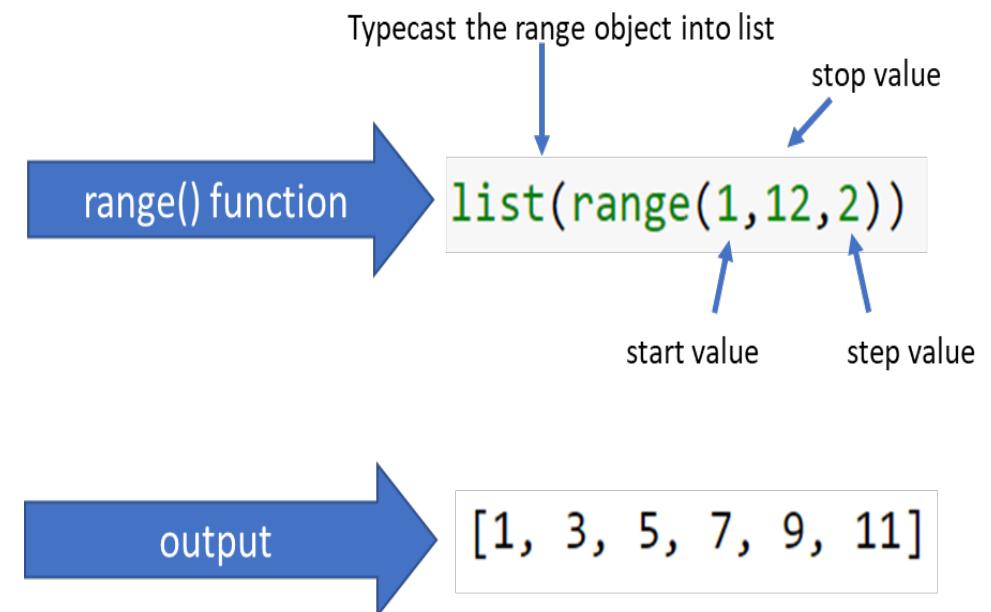
Program to generate sequence of numbers 1 to 4.



# Range

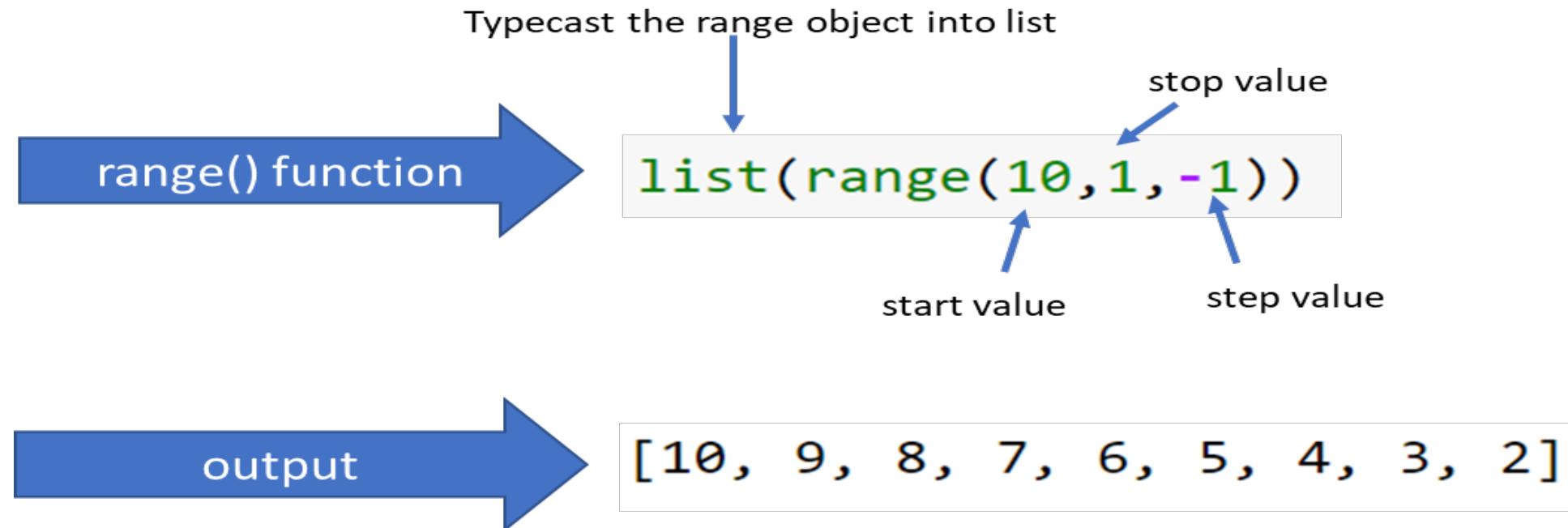
- Range() with three arguments:
- It has **start value**, **stop value** and the **step value**, all three values are given inside range () .
- Here first **start value** is 1, **stop value** 12 and **step value** is 2. That's why the output is [1, 3, 7, 9, 11].

Program to generate a **sequence** of **alternate natural number** starting from 1 to 12.



# Example

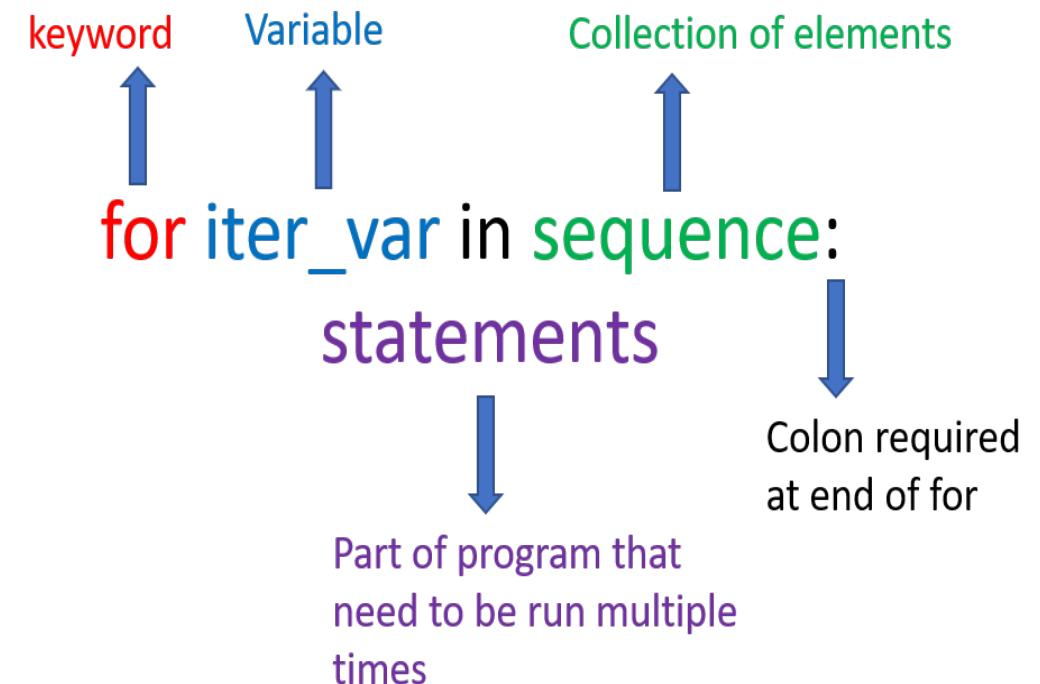
- Write a program to generate a sequence of starting from 10 to 2.



# For loop

- ❑ For loop is used when we want to run a part of our program multiple times.
- ❑ It is also used to traverse sequences in python like lists, tuple etc.

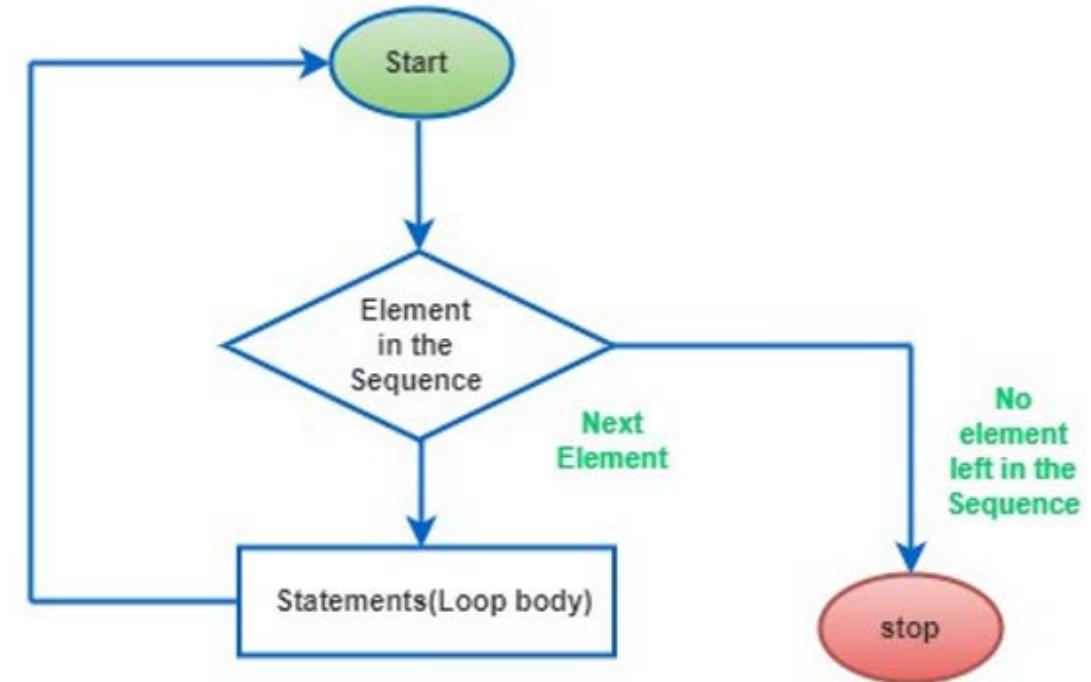
## Syntax of For loop:



# For loop:

## Flowchart of For loop:

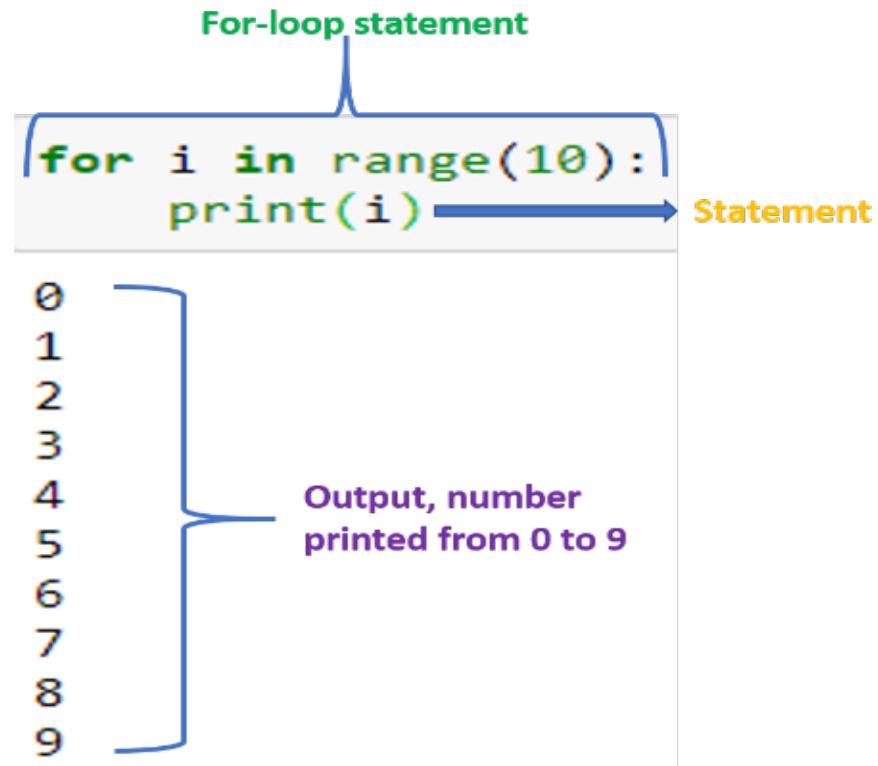
- In this flowchart the statements which we want to run **multiple times** would keep on running till, we have **elements** in the sequence.
  
- As no **element** left, it will come **out from loop**.



# Example:

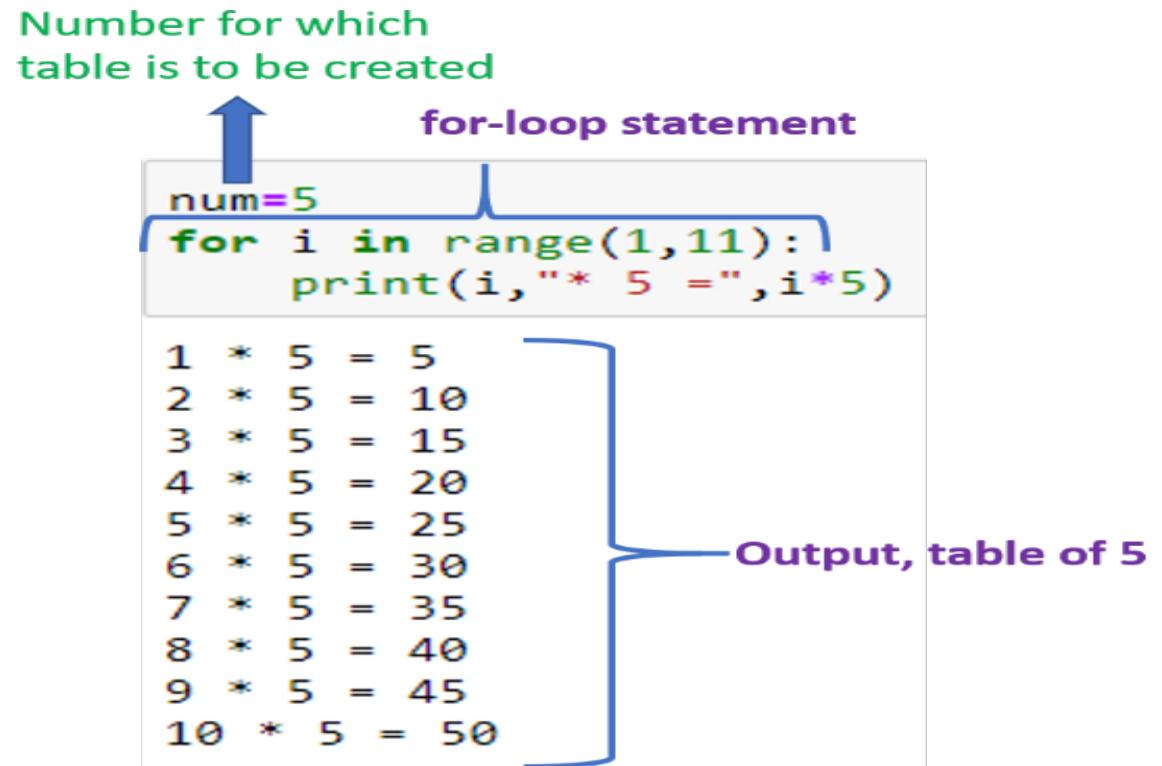
## Example1:

For loop with range function:



## Example2:

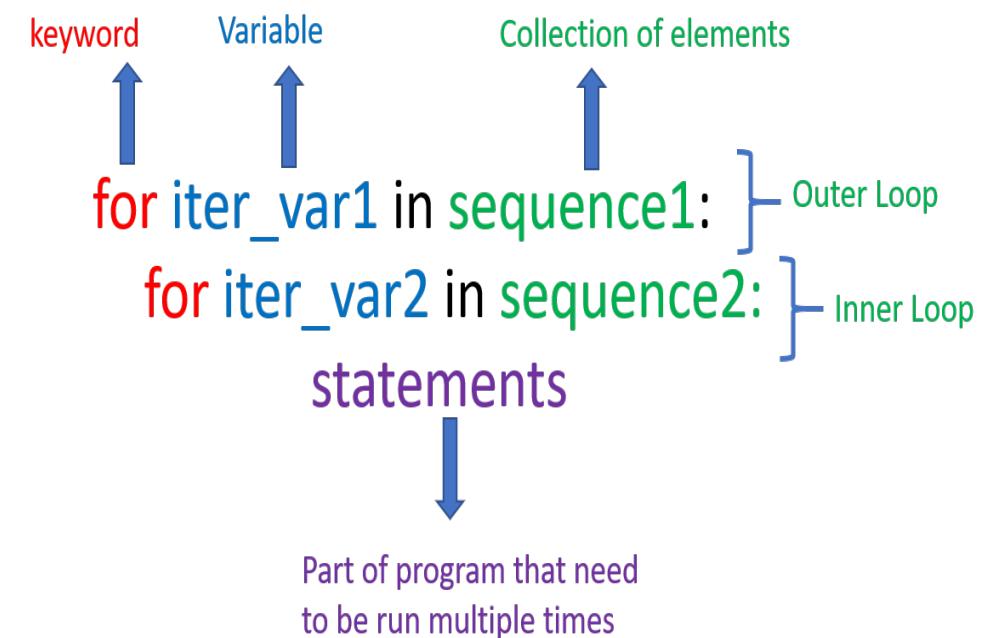
Table of number 5 is printed using for loop



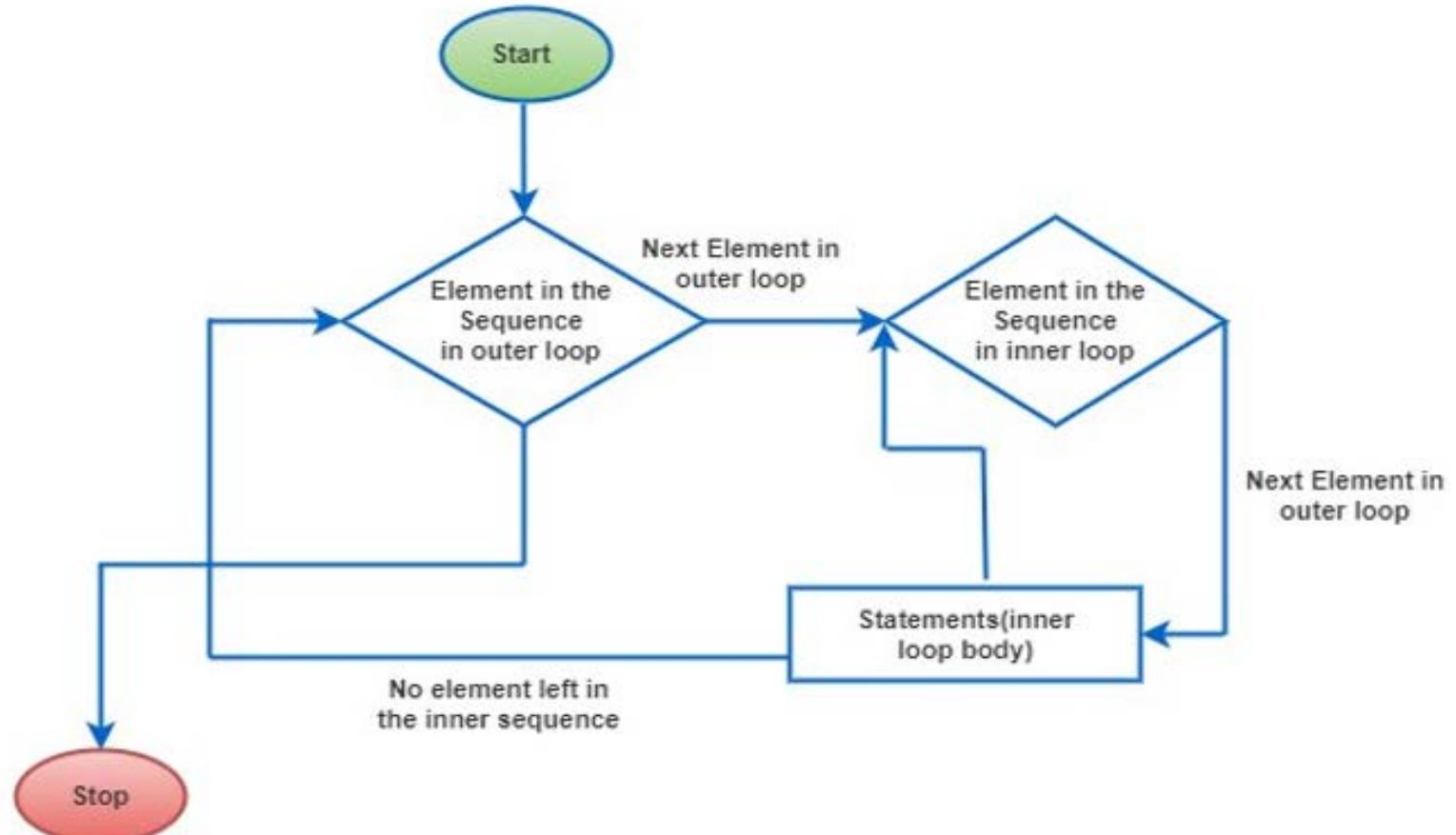
# Nested For loop:

- Nested for loop is called when we use for loop inside a for loop.
- In python we can use any number of for loop inside a for loop.
- Generally, we call main for loop as outer for loop and nested for loop as inner for loop.
- If outer loop running m times and inner loop running n times, then total iterations would be  $m*n$ .

## Syntax of Nested For loop:

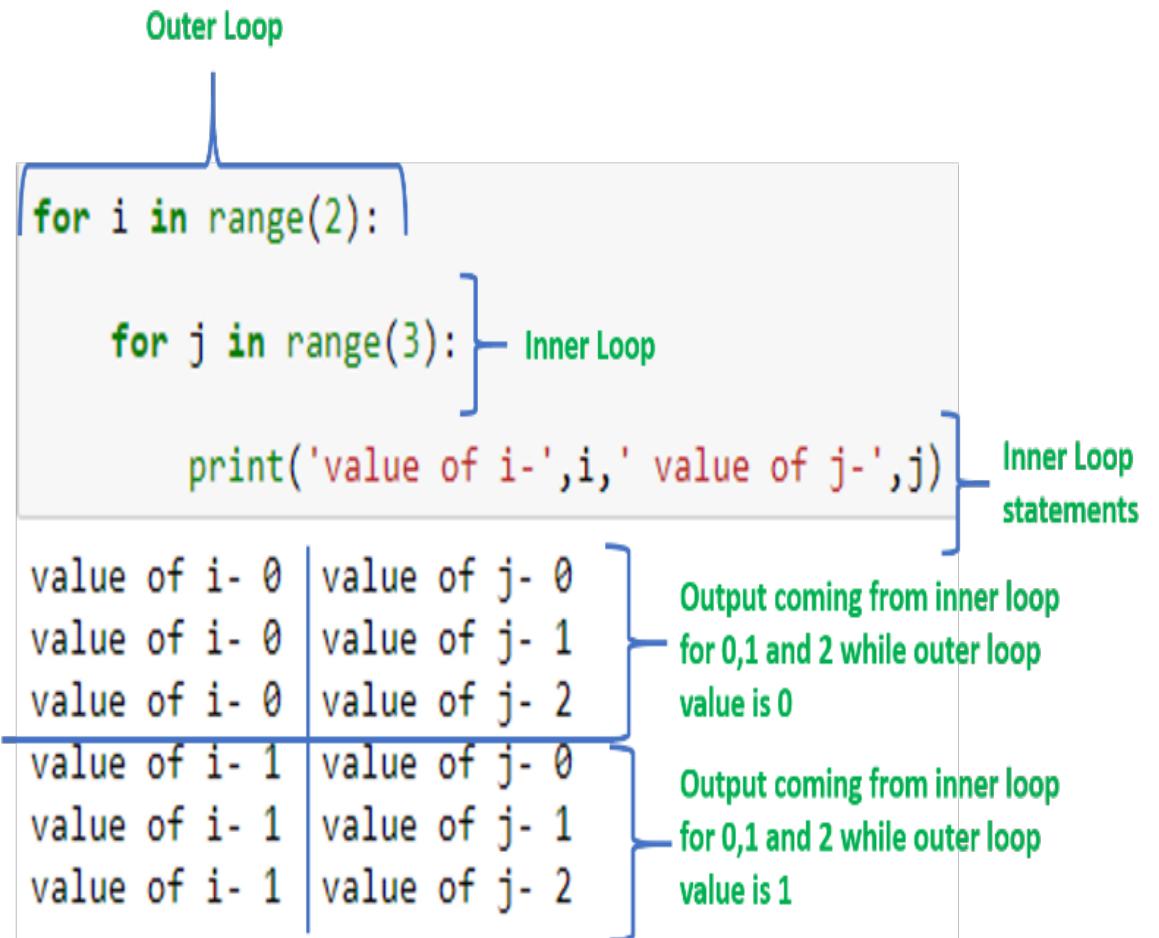


# Flowchart of Nested For loop:



# Example

- We have used two **for loops**.
- Outer loop is running for *two values*-0 and 1 and inner loop is running for *three values*-0,1 and 2 and total print statements we are getting  $2*3=6$ .



# Can you answer these questions?

1. What is the **output** of the following **code snippet**??

a.

```
string3 = 'Technology'
for i in range(len(string3)):
    string3[i].upper()
print (string3[4])
```

b.

```
num1 = 5
num2 = 4
while ( num2>=1):
    print ('&')
    for index in range (1,num1+1):
        print ('+')
    num2= num2-1
    print ('*')
```

- A. N ←
- B. n
- C. H
- D. h

How many time special character print?

- A. 11 ←
- B. 8
- C. 10
- D. 6

3. Which of the below programs will print all integers that aren't divisible by either 2 or 3 and lies between 1 and 50?

A.

```
for i in range(0,50):
    if(i%4!= 0 and i%6!=0):
        print(i)
```

B.

```
for i in range(0,51):
    if(i%4!= 0 and i%6!=0):
        print(i)
```

C.

```
for i in range(0,51):
    if(i%4!= 0 and i%3!=0):
        print(i)
```

D.

```
for i in range(0,51):
    if(i%2!= 0 and i%3!=0):
        print(i)
```



# Summary:

- ❑ Range function () returns a sequence of numbers.
- ❑ A **for** loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).
- ❑ Nested **for loop** allows us to create one for loop inside another for loop.

# Session Plan - Day 5

## 2.4 Jump statements

- Break
- Continue

## 2.5 Else With Loop

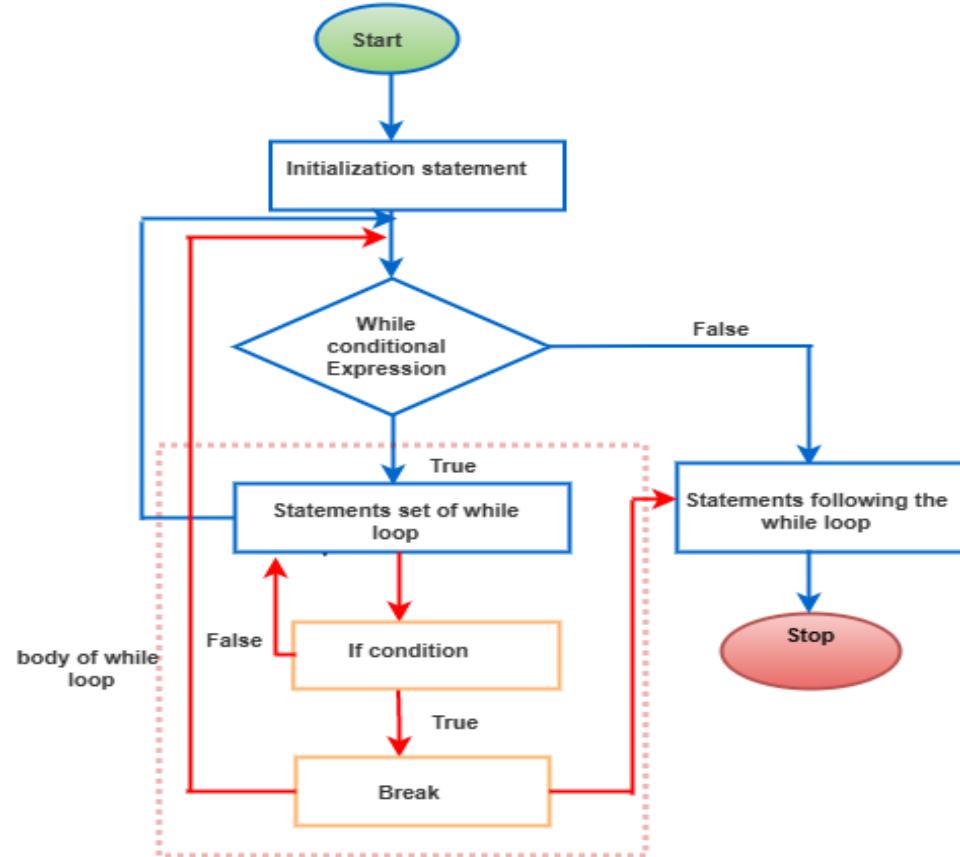
- While with else statement
- For with else statement
- Examples
- Review questions
- Summary

# Jump Statements

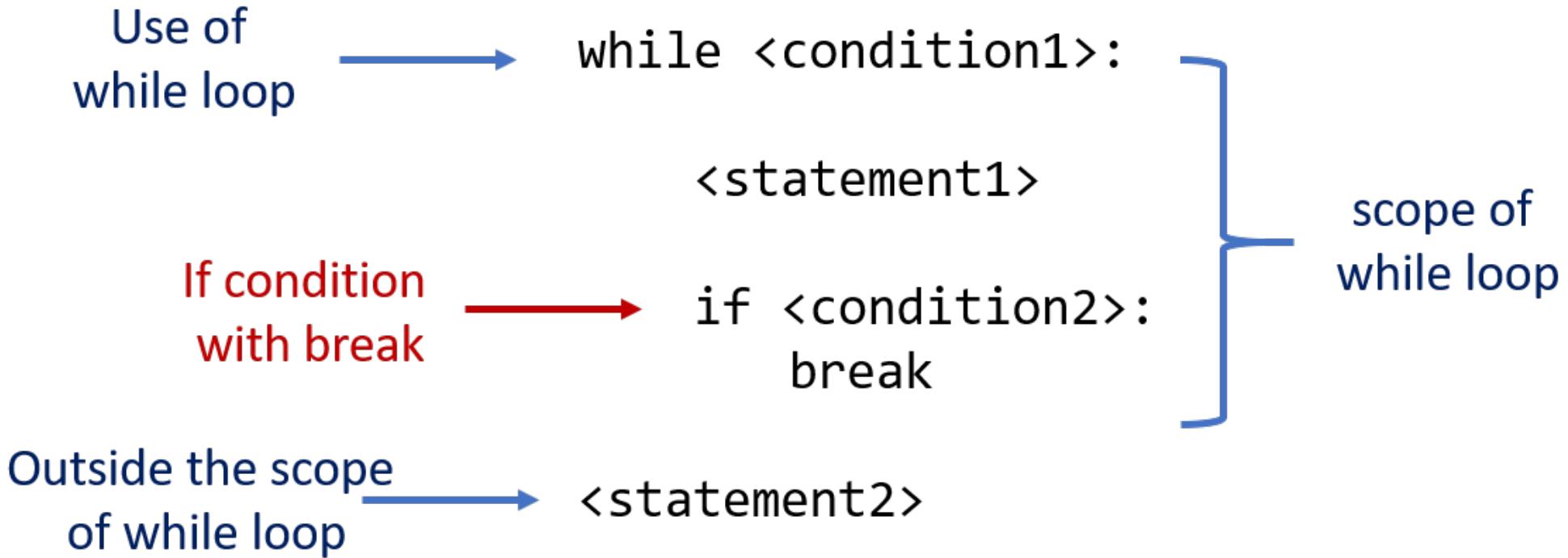
- It is used to :-
  - Jump, skip or terminate the iteration or loop from the running program
  - Interrupt loops
- Also known as early exit from loop.
- Jump statements are of two types:-
  - Break
  - Continue

# Break

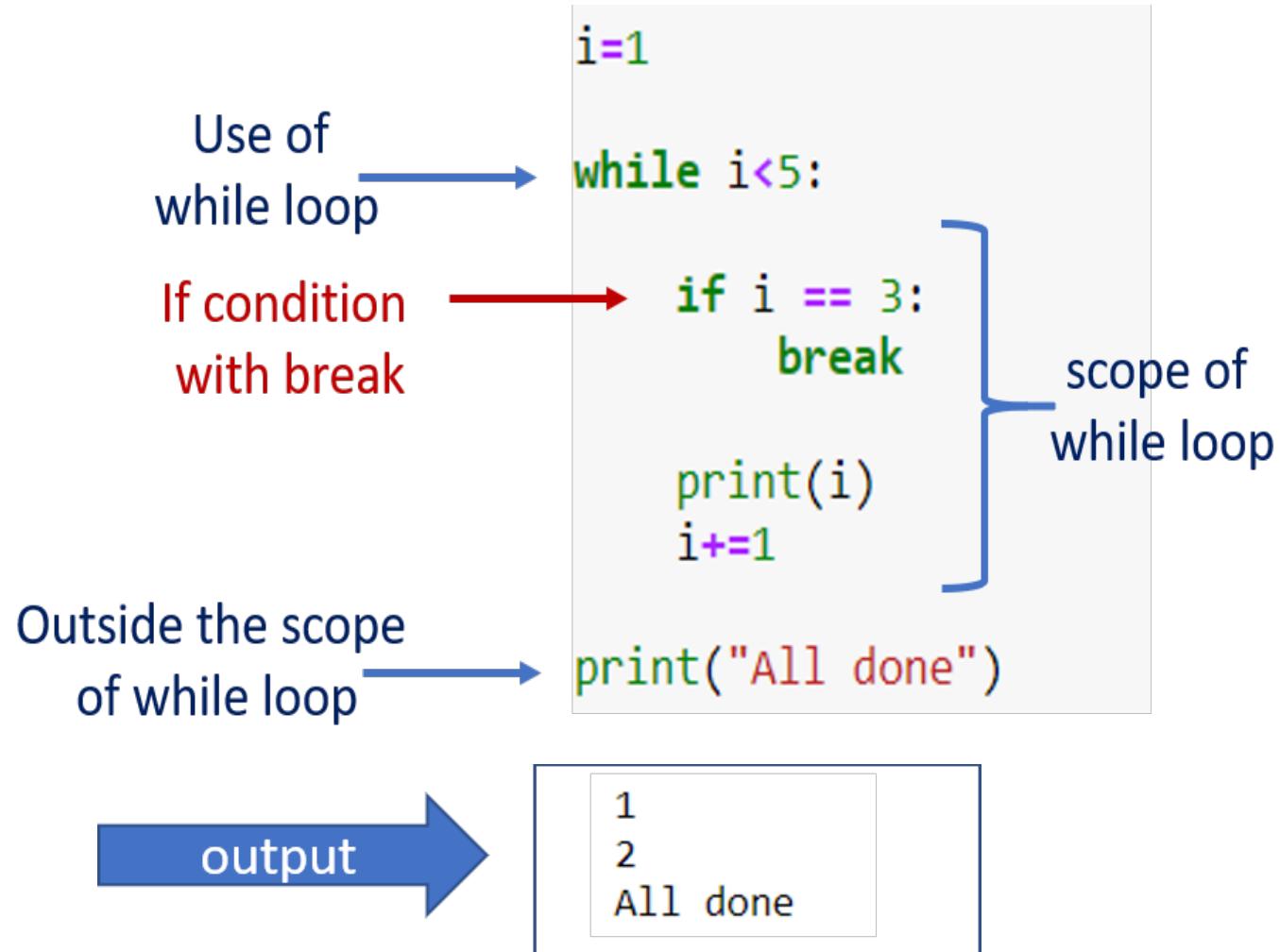
- It terminates the execution of loop immediately
- Execution will jump to the next statements.



# Syntax



# Example

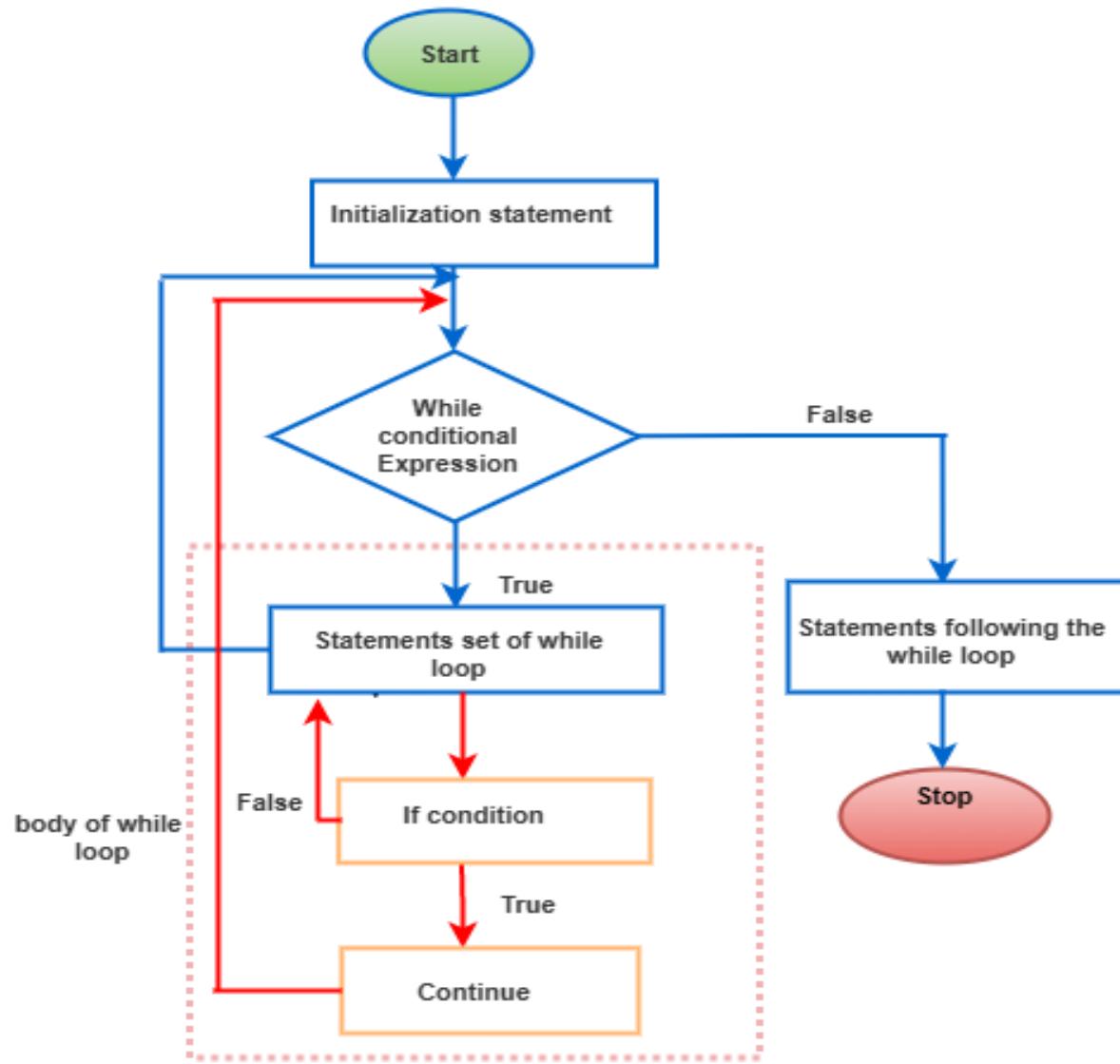


## Explanation :-

- when value of I would be 3 break statement will be executed and loop will be terminated and control moves outside the while loop to print “All done”.
- If there would have no break, then loop would had run for i=1 to 4.

# Continue

- It is used to :-
- Skip ongoing iteration
- continues remaining iterations of loop.
- Jump to the next iteration by skipping the existin



# Syntax

Use of while loop → `while <condition1>:  
 if <condition2>:  
 continue  
 <statement1>  
 <statement2>`

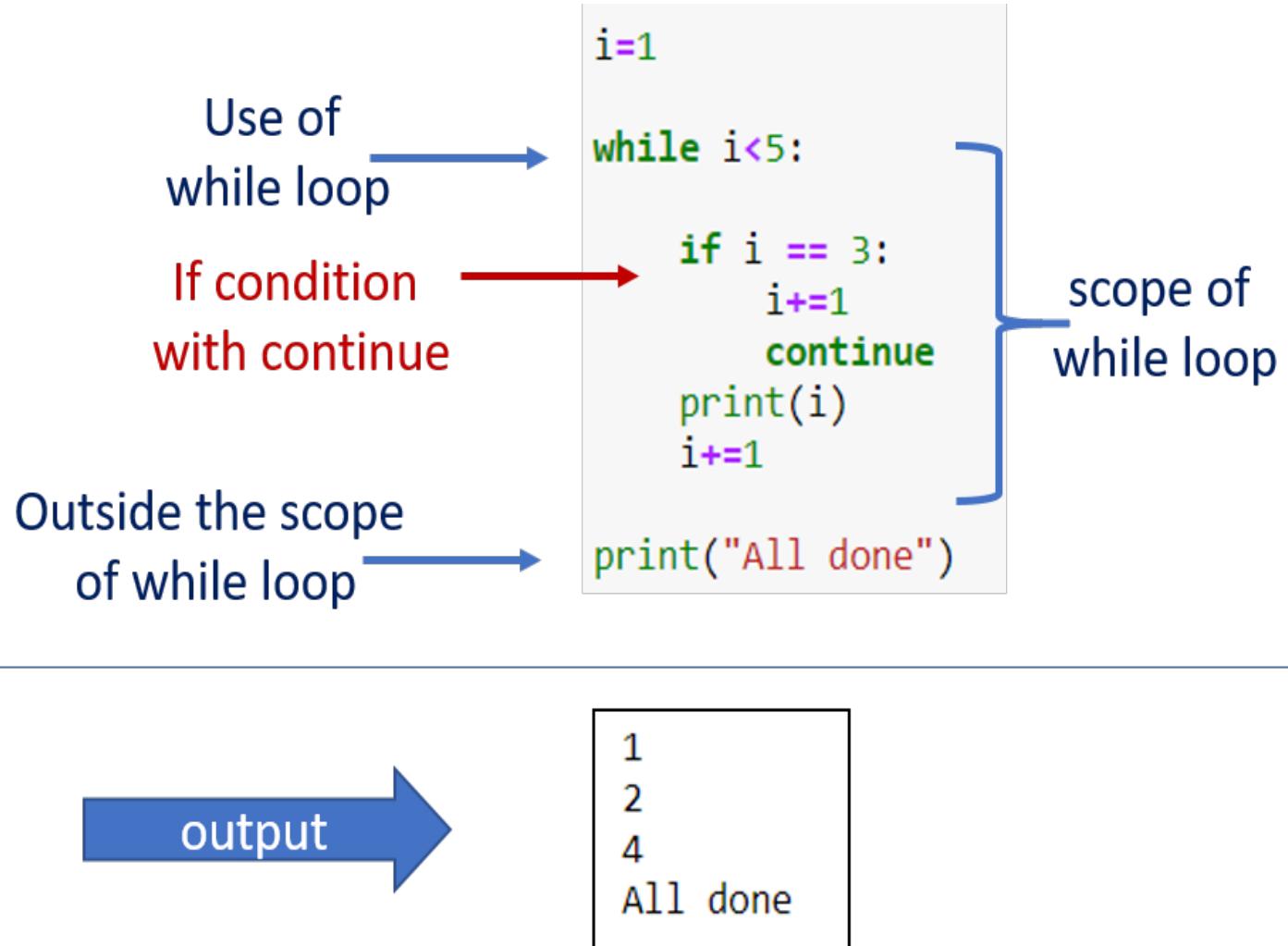
If condition with continue → `<statement3>`

Outside the scope of while loop → `<statement3>`

scope of while loop

Note: In break statement, the whole loop will end; and in continue statement, only the specific iteration will end.

# Example



## Explanation:-

- The continue statement will interrupt the on-going iteration for  $i=3$  and skip all the statements of current iteration which are mentioned after the continue statement like `print(i)`.
- The control goes back to the while loop for next iteration and loop moves on.

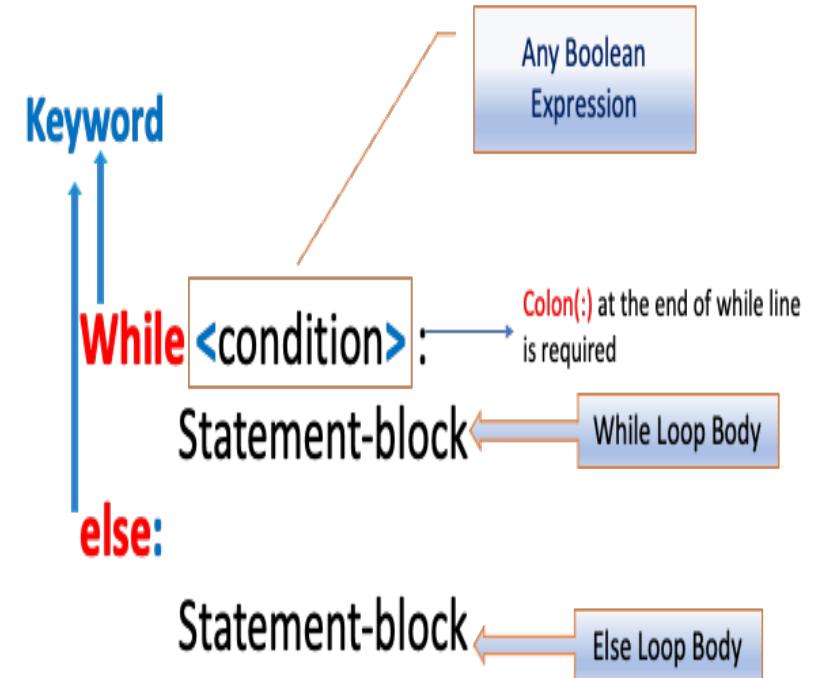
# Else with Loop

- ❑ While with else statement
- ❑ For with else statement

# While with else Statement

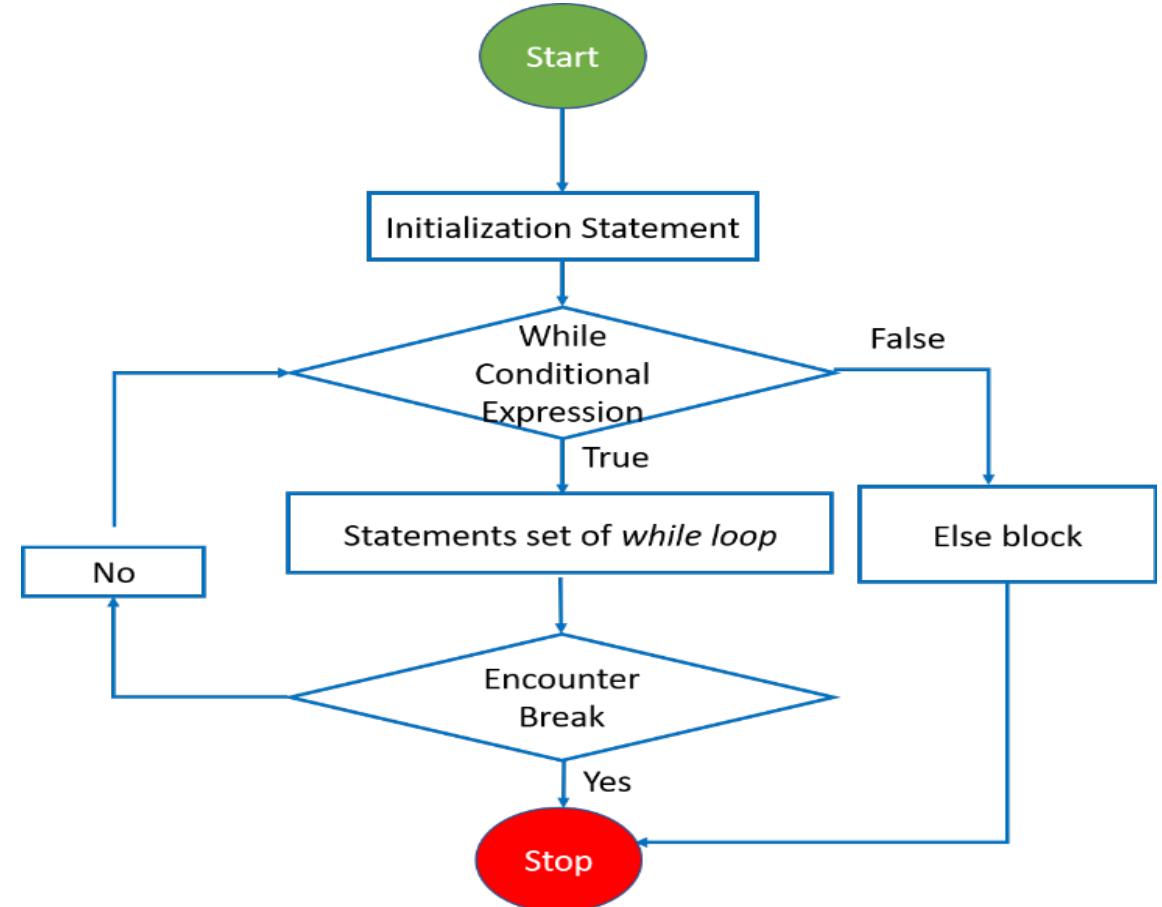
In python the while statement may have an exceptional else clause.

- ❑ If the condition in the while loop evaluates to False, the else portion of the code runs.
- ❑ if break statement is used in while loop then else section is not taken into consideration.



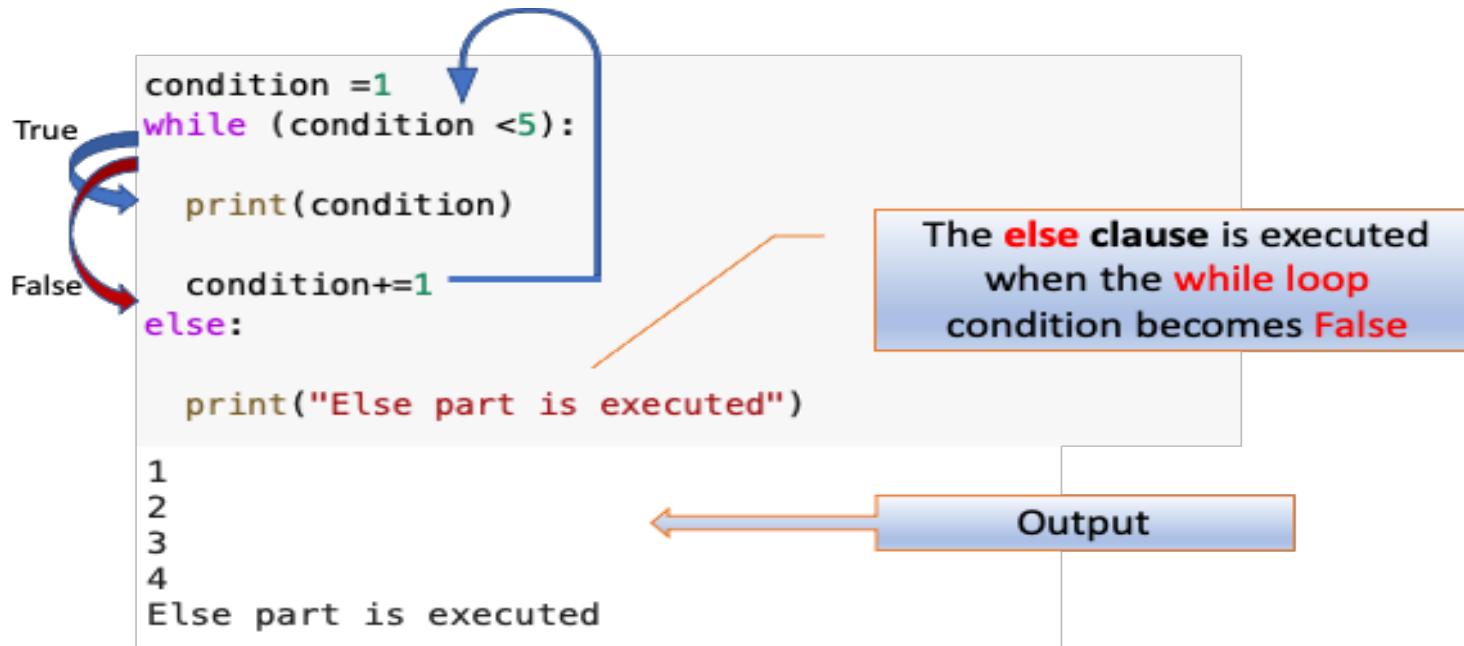
# Flowchart of While with else loop

- The else clause will be executed when the condition becomes False and the loop runs normally.
  
- The else clause, on the other hand, will not execute if the loop is terminated early by a break or return statement.



# Example

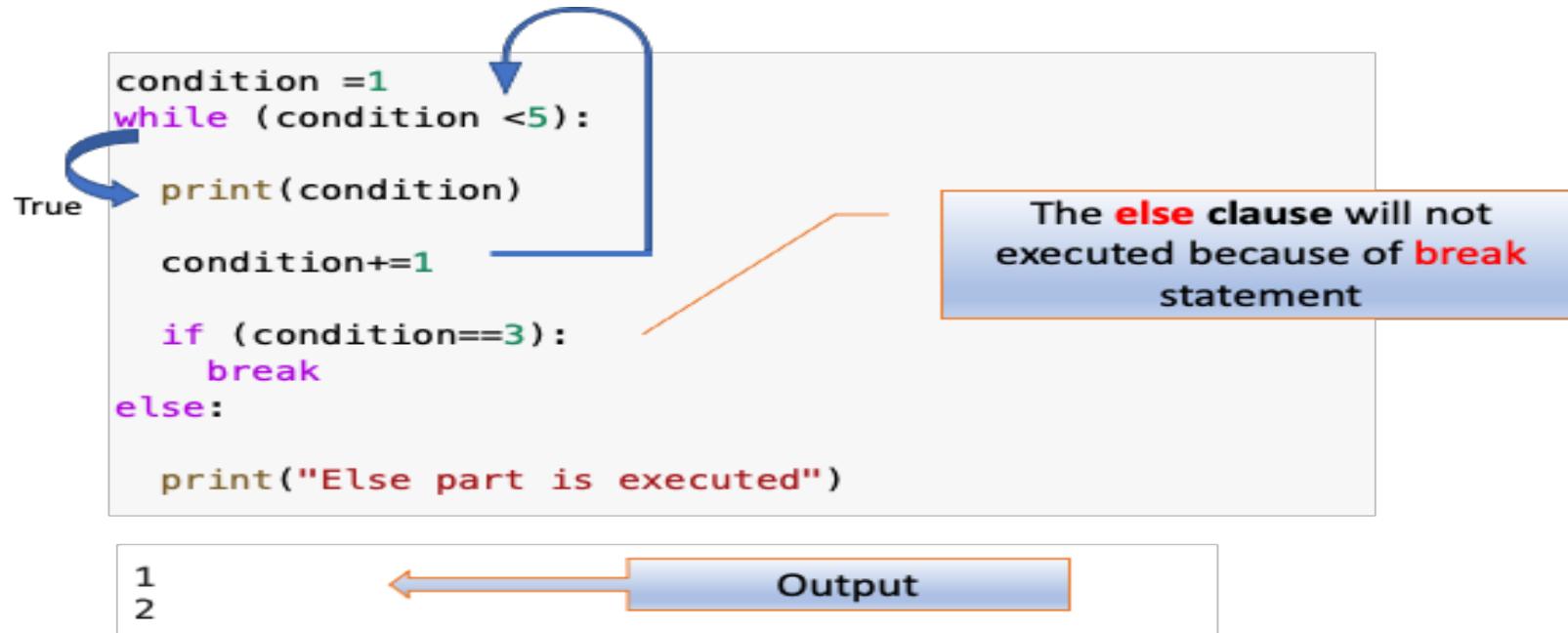
## Without Break Statement



- Without Break Statement
- While loop will execute normally up to a given condition. No early exit is there so else block will be executed.

# Example

## □ With Break Statement

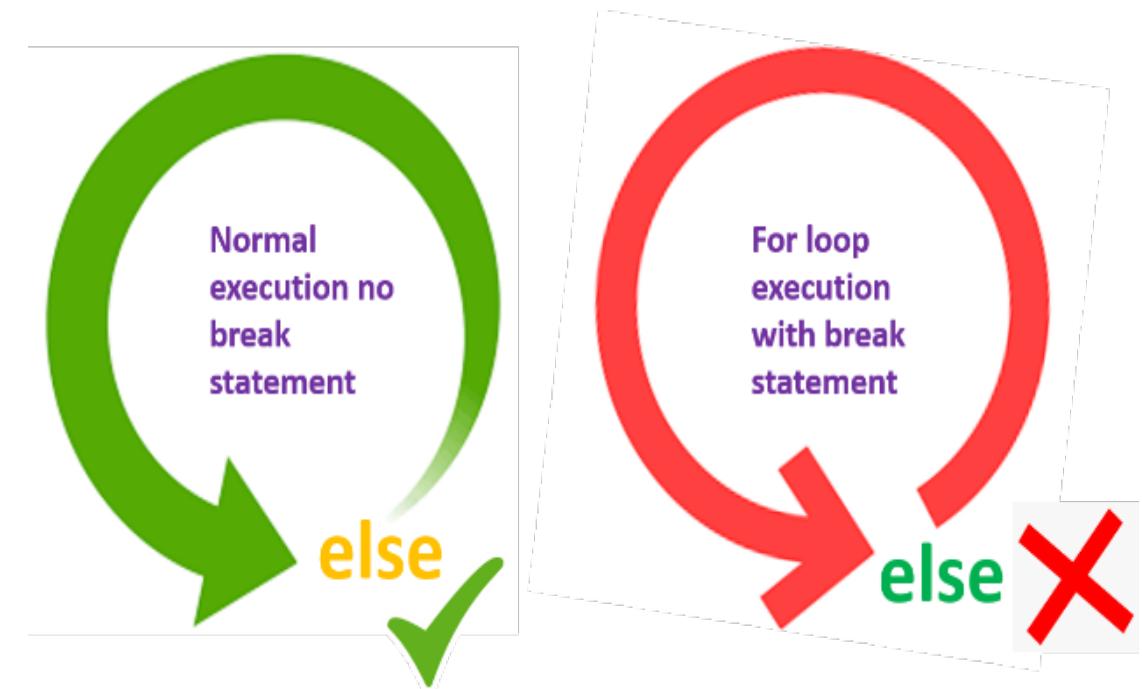


- While loop will get terminated when value of condition variable = 3, So else block will not be executed.

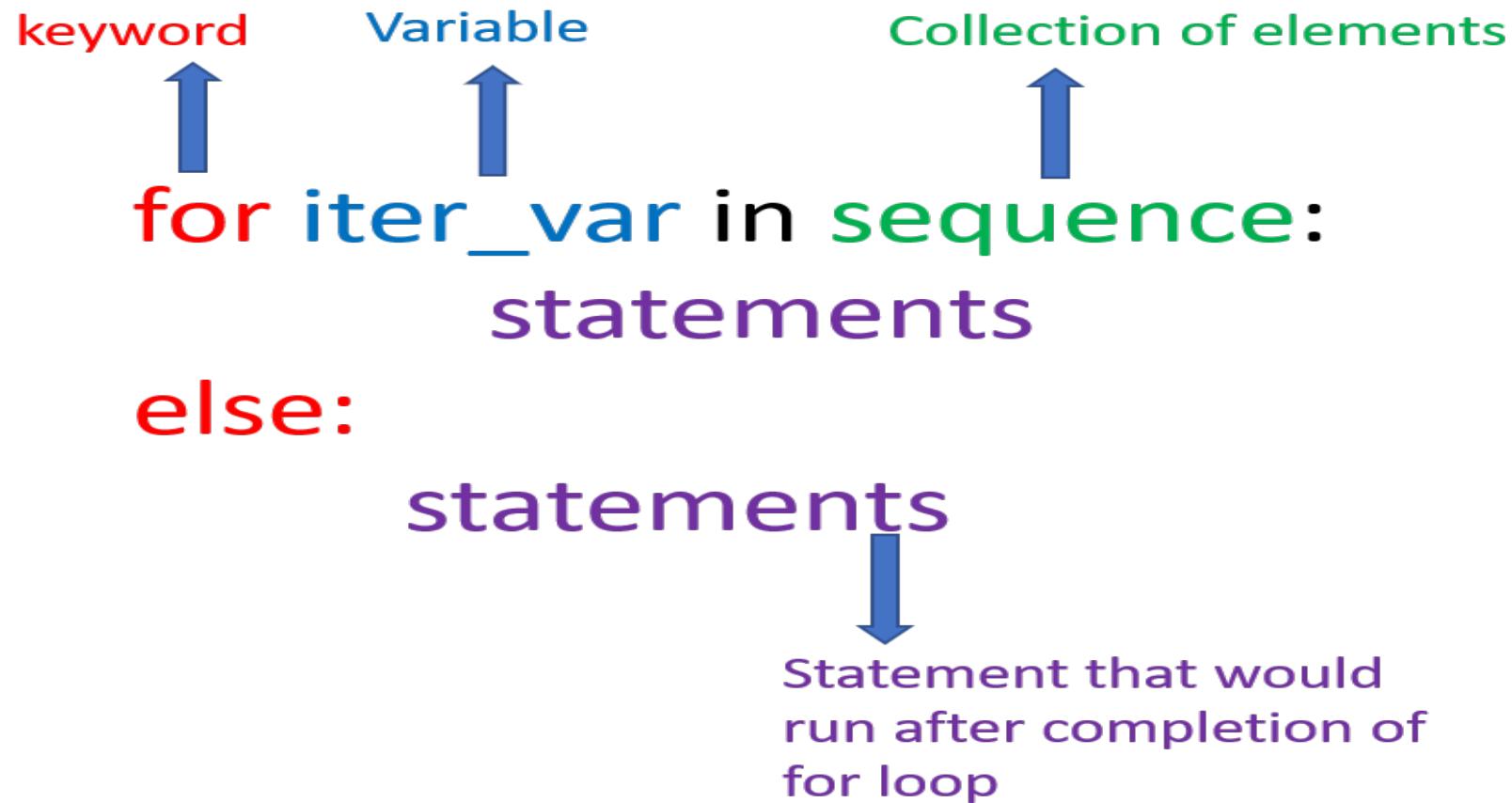
# For with else statement

Python allows **else with for loop**.

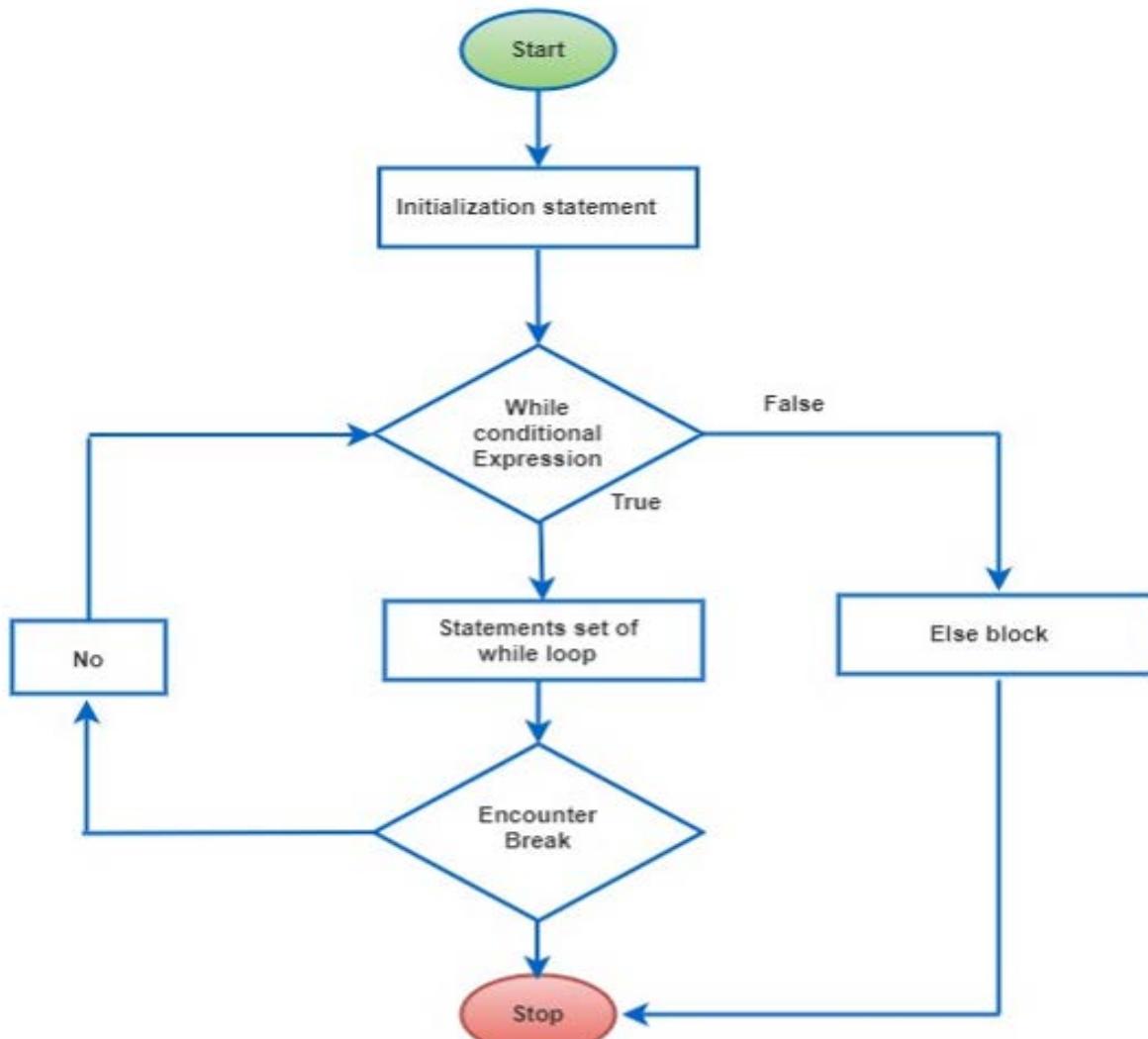
- When all the iteration of **for loop** are finished normally, then control goes to **else** part.
  
- If loop has a break statement, then **else** part would not execute.



# Syntax of *for with else*



# Flow chart of *for* with *else*



# Example

- Program to demonstrate for loop with else.

```
for i in range(5):  
    print(i)  
else:  
    print("Loop finished, i am in else part")
```

0  
1  
2  
3  
4

Output from for-loop

Loop finished, i am in else part

for-loop with statement

Body of else

Else statement executing after for loop execution finished

- For loop has been used with else, we can clearly see that, first all the elements of sequence get executed and when all values from 0 to 4 gets printed, control goes to else part of the program.

# Example

- Program to demonstrate for loop with break and else.

```
for i in range(11): ]— for loop with sequence
    if(i==5):
        break ]— Break with condition for i=5
    print(i)
else:
    print("Loop finished, i am in else part") ]— else body
```

0  
1  
2  
3  
4

Output from 0 to 4 then break executed and  
program terminated and no else part executed

- We have used break statement with the condition for  $i=5$ , so in the output we are getting values from 0 to 4 and then no else part executed.



# Example

- Write a program to search a given color name in list of colors. Use Linear/Sequential Search Techniques.

```
color_name = ['Red', 'Green', 'Blue', 'Black', 'Pink']
name = input("Enter the color name to be search ")
for c in color_name:
    if name==c:
        print(f"Color name {c} found in list")
        break
else:
    print("Color name not found in list")
```

**Test case - 1** For the input value ‘Blue’, output will be –

Enter the color name to be search Blue  
Color name Blue found in list

**Test case - 2** For the input value ‘Yellow’, output will be –

Enter the color name to be search Yellow  
Color name not found in list

# Example

- Write a program to find a given number is prime or not. A number is prime number if it is divisible only by itself and 1.

Approach:

- Every number is divisible by 1 and itself, so in the below code we check that a given number is divisible by 2 to n-1 or not.
- If divisible means not a prime number otherwise a prime number.

```
n = int(input("Enter any positive number"))
for i in range(2,n):
    if n%i==0:
        print(f"{n} is not a prime number")
        break
else:
    print(f"{n} is a prime number")
```

**Test case - 1 For the input value 11, output will be –**

**Enter any positive number11  
11 is a prime number**

**Test case - 2 For the input value 12, output will be –**

**Enter any positive number12  
12 is not a prime number**

# Can you answer these questions?

1:-what will be the output of following code:-

```
for i in range(10):
    print(i)
    if(i == 7):
        print('break')
        break
```

- A) 0  
1  
2  
3  
4  
5  
6  
7  
**BREAK**
- B) 0  
1  
2  
3  
4  
5  
6
- C) 0  
1  
2  
3  
4  
5  
6  
7
- D) **NONE**
- 

2:- What will be the output of following code:-

```
for i in range(6):
    if(i==3):
        continue
    print(i)
```

A) 0  
1  
2  
3  
4  
5  
**continue**

B) 0  
1  
2  
4  
5

C) 0  
1  
2  
3  
4  
5  
6  
**continue**

D) **NONE**



# Summary

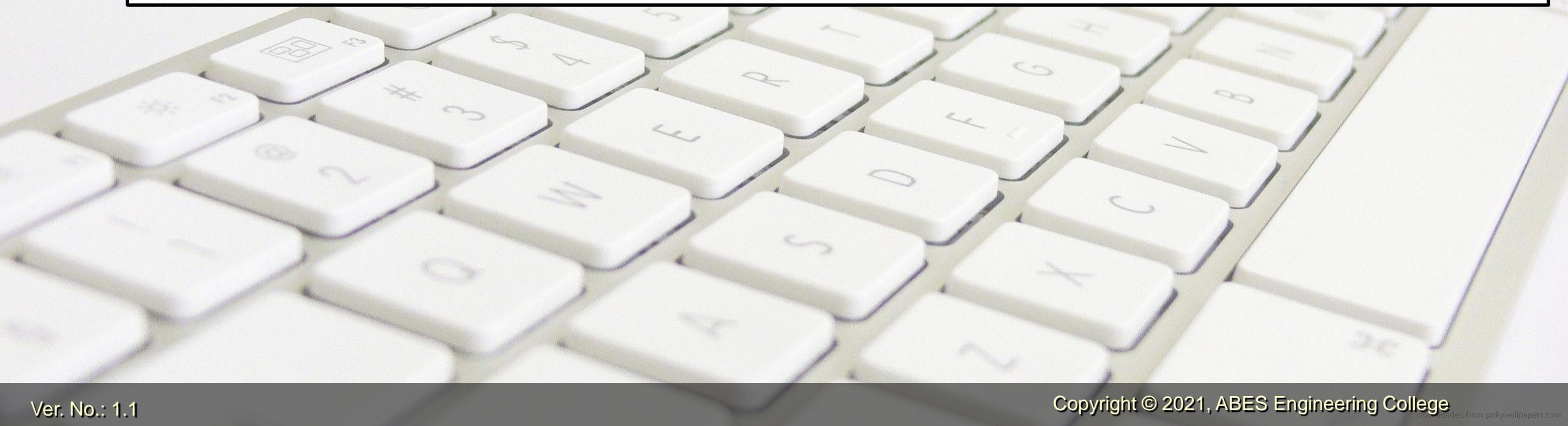
- Jump statements in python are used to alter the flow of a loop like you want to skip a part of a loop or terminate a loop.
- Break Statement in Python is used to terminate the loop.
- Continue Statement in Python is used to skip all the remaining statements in the loop and move controls back to the top of the loop.

# References

- ❑ <https://docs.python.org/3/tutorial/controlflow.html>
- ❑ Think Python: An Introduction to Software Design, Book by Allen B. Downey
- ❑ Head First Python, 2nd Edition, by Paul Barry
- ❑ Python Basics: A Practical Introduction to Python, by David Amos, Dan Bader, Joanna Jablonski, Fletcher Heisler
- ❑ <https://fresh2refresh.com/python-tutorial/python-jump-statements/>
- ❑ <https://tutorialsclass.com/python-jump-statements/>

# Thank You

# 3.Python Collections and Sequences



# General Guideline

© (2021) ABES Engineering College.

This document contains valuable confidential and proprietary information of ABESEC. Such confidential and proprietary information includes, amongst others, proprietary intellectual property which can be legally protected and commercialized. Such information is furnished herein for training purposes only. Except with the express prior written permission of ABESEC, this document and the information contained herein may not be published, disclosed, or used for any other purpose.

# Objective of Collections/Sequences

To describe the importance of list , string, tuple , dictionary and set in python

To uses to store multiple data values.

To use the index to update, add, and remove items

To use the data structures with out indexes

To explain the difference among different Collections

To select collections built-in functions in Python to write programs in Python.

# Topics Covered

Day 1

## 3.1 Introduction

- 3.2 String
  - Creation of String
  - Accessing the String

Day 2

## 3.2 String

- Updation
- Deletion
- Built-in-methods
- Operations
- String Formatters
- Loops with Strings

Day 3

## 3.3 List

- Creation
- Accessing
- Update
- Built in methods

Day 4

## 3.3 List

- Loops
- Nested List
- List Comprehensions

# Topics Covered

Day 5

## 3.4 Tuple

- Creation
- Accessing
- Modification
- Built in Methods
- Operations

Day 6

## 3. 5 Dictionary

- Creation
- Accessing
- Modification

Day 7

## 3.5 Dictionary

- Built in Methods
- Loops and Conditions

Day 8

## 3.6 Set

- Creation
- Assessing
- Modification
- Built in methods
- Operators
- Loops
- Frozen set

# Session Plan - Day 1

## 3 Introduction to Python Collections and Sequences

### 3.1 String

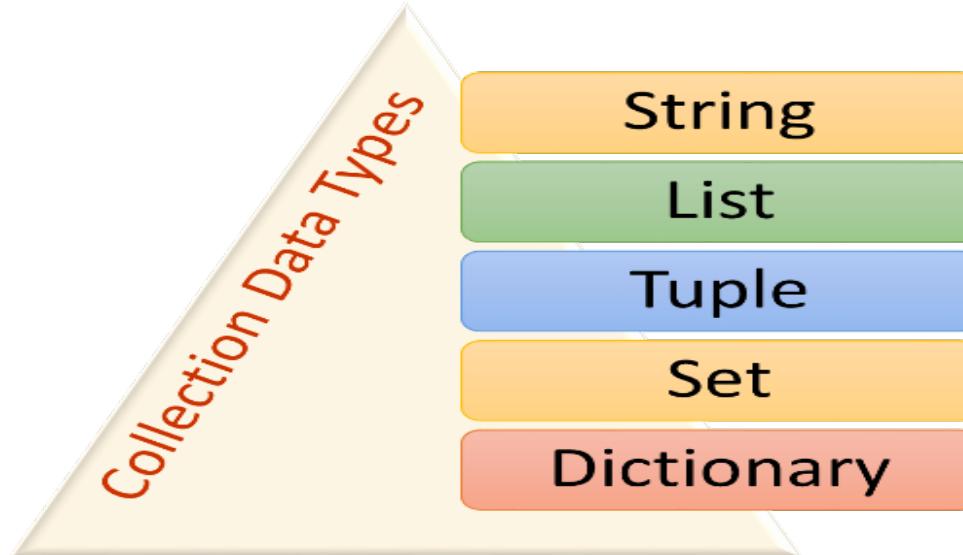
- **Creation of String**
- **Assessing of String**
- **Updation of String**
- **Examples**
- **Review Questions**

# Introduction

In **real life** we need to store data like-

- Name of students** – A sequence of alphabet character
- Marks of n students** – A sequence of either integer or float value
- Student's record** – A sequence of name, branch, roll number, address etc

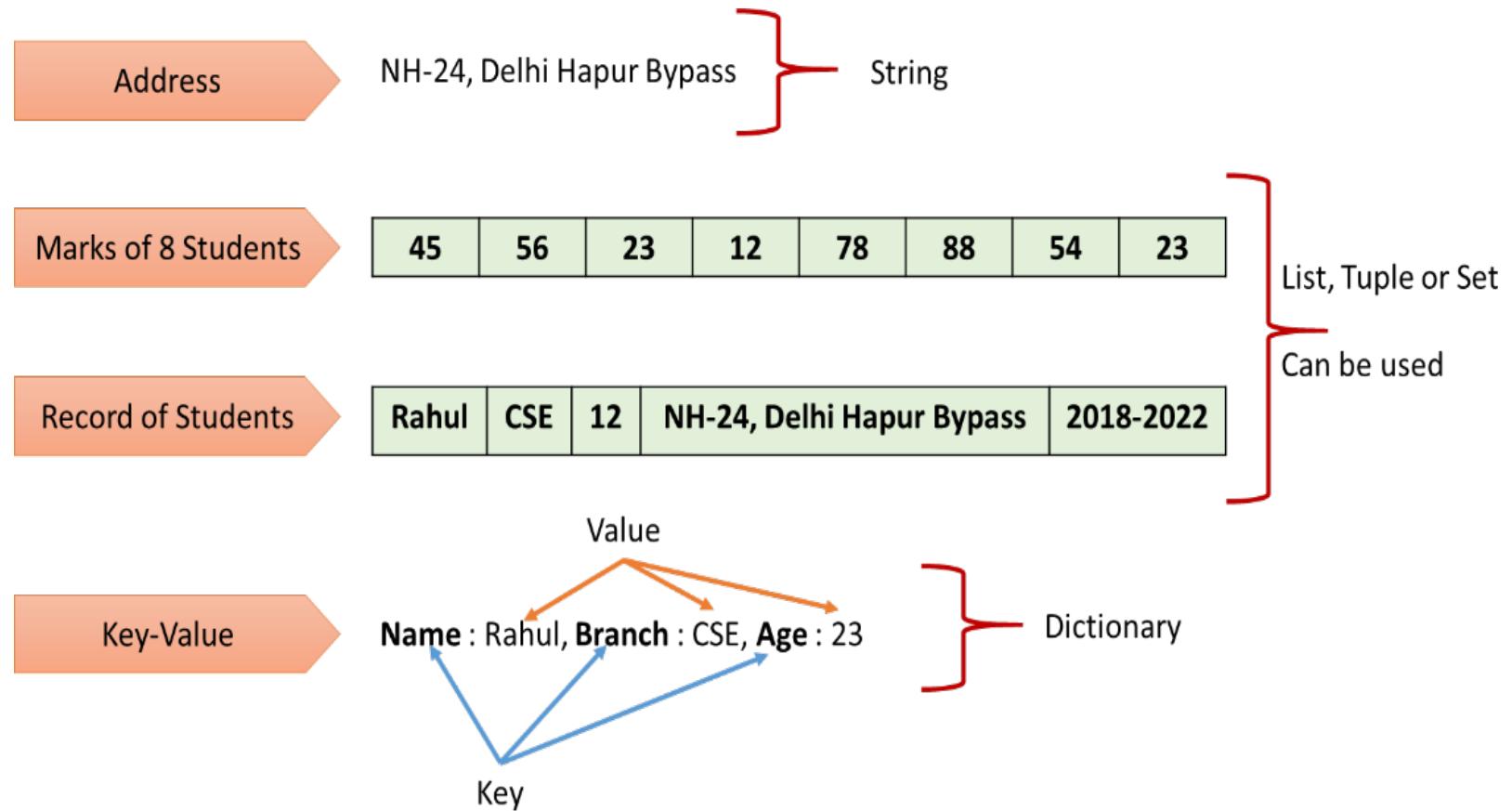
So to represent these type of data in python provides us some built in collections.



# Contd...

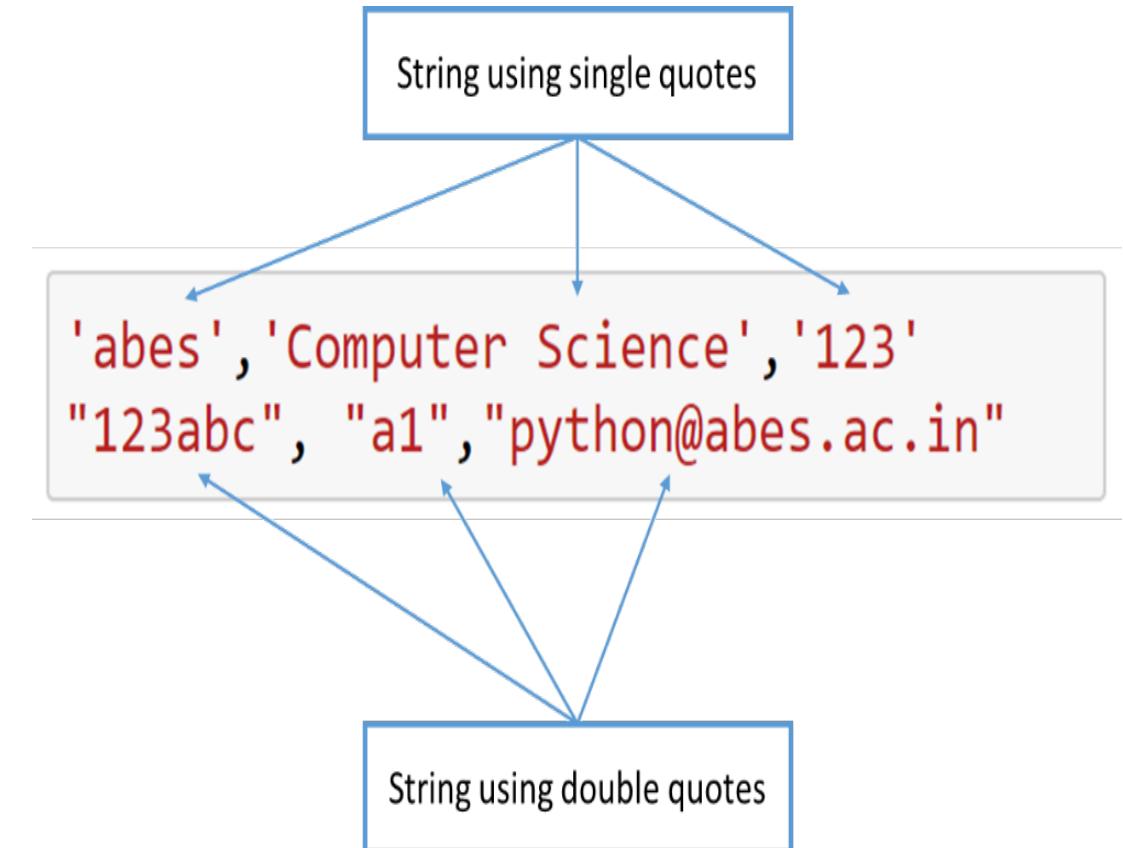
## Representation of Python Collections with the help of Figure:

- If data is the combination of numeric data, alphanumeric or of different types than we go for List, Tuple or set .



# String

- A string is a sequence of alphanumeric and special character.
- String is the collection of characters, it may compose of alphanumeric and special characters.
- Strings are created in many ways using single quotes or double-quotes.



**Alphanumeric characters:** a-z, A-Z, 0-9; **Special symbol:** \*, -, +, \$, @, whitespace .... etc

# Creation of Empty String

## Empty String

- An empty string is a string having no character.
- We have three ways to create an empty string as shown in syntax.
- An Empty String is created by single quotes, double quotes and str().



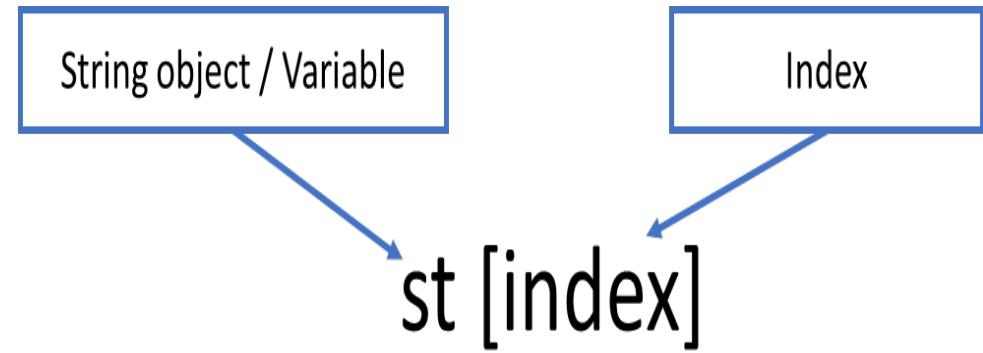
# Creation of Non Empty String

- A non-empty string is a sequence of alphanumeric character with at least one character.
- A non-empty string is created by using single quotes and double quotes

```
st1 = 'abes' #String creation using single quotes
st2 = "abes" #String creation using double quotes
```

# Assessing the String

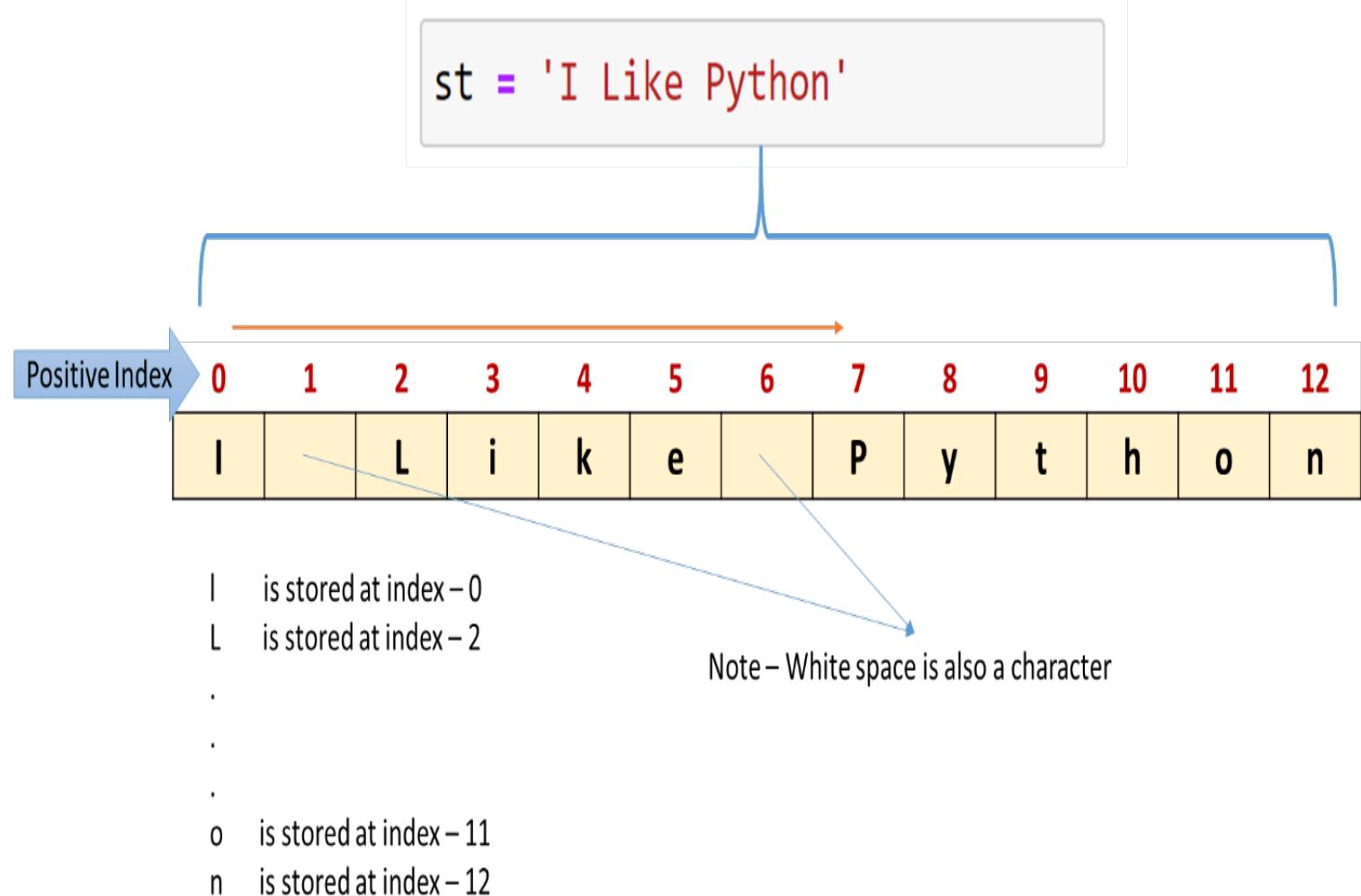
- String Element can be accessed using Square Bracket.
- These Square brackets [ ] take an index as input.
- In Python we have two types of Indexing.
  - ❖ Positive Indexing
  - ❖ Negative Indexing



# Positive Indexing

- Positive indexing starts from Zero (0) and from left hand side i.e. first character store at index 0, second at index 1 and so on.

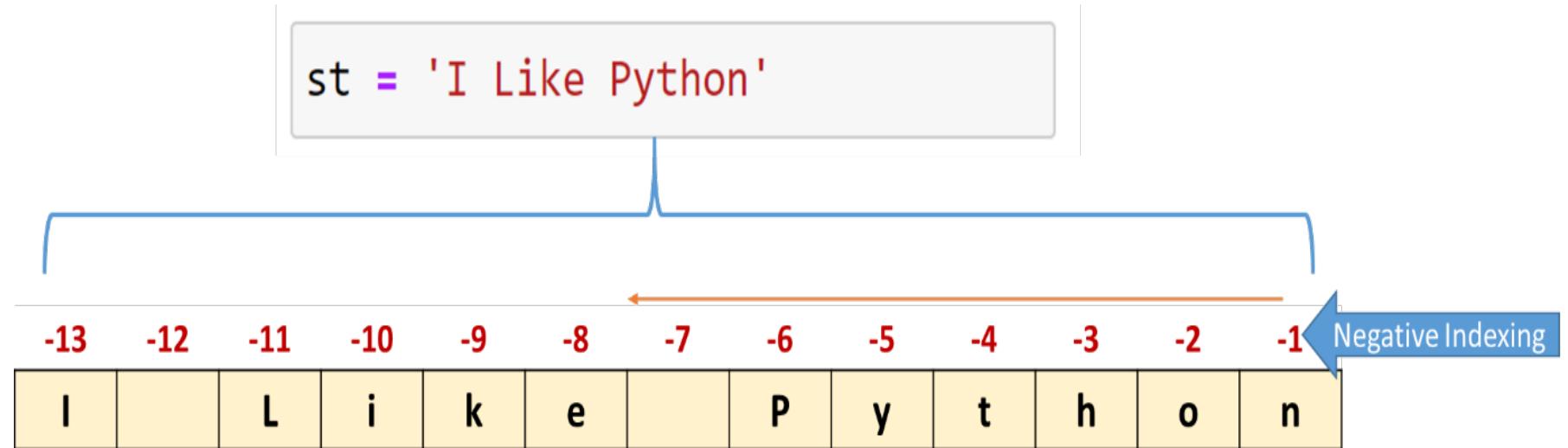
- Every character of string has some index value.



# Negative Indexing

## □ Negative

Indexing starts from negative indexing start from -1 and from right-hand side.



## □ Last character store at index -1 and as we move towards the left, it keeps increasing like -2, -3, and so on.

n is stored at index → -1

o is stored at index → -2

.

.

.

I is stored at index → -13

# Example

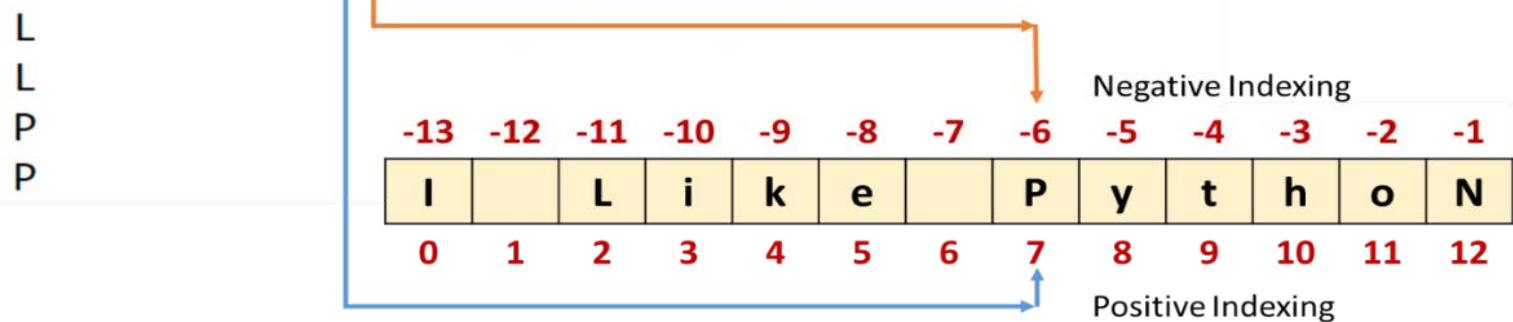
- Write a program to print character 'P' and 'L' using positive indexing and negative indexing.

Assume a string **st = 'I Like Python'**, is given.

```
st = 'I Like Python' # A given string

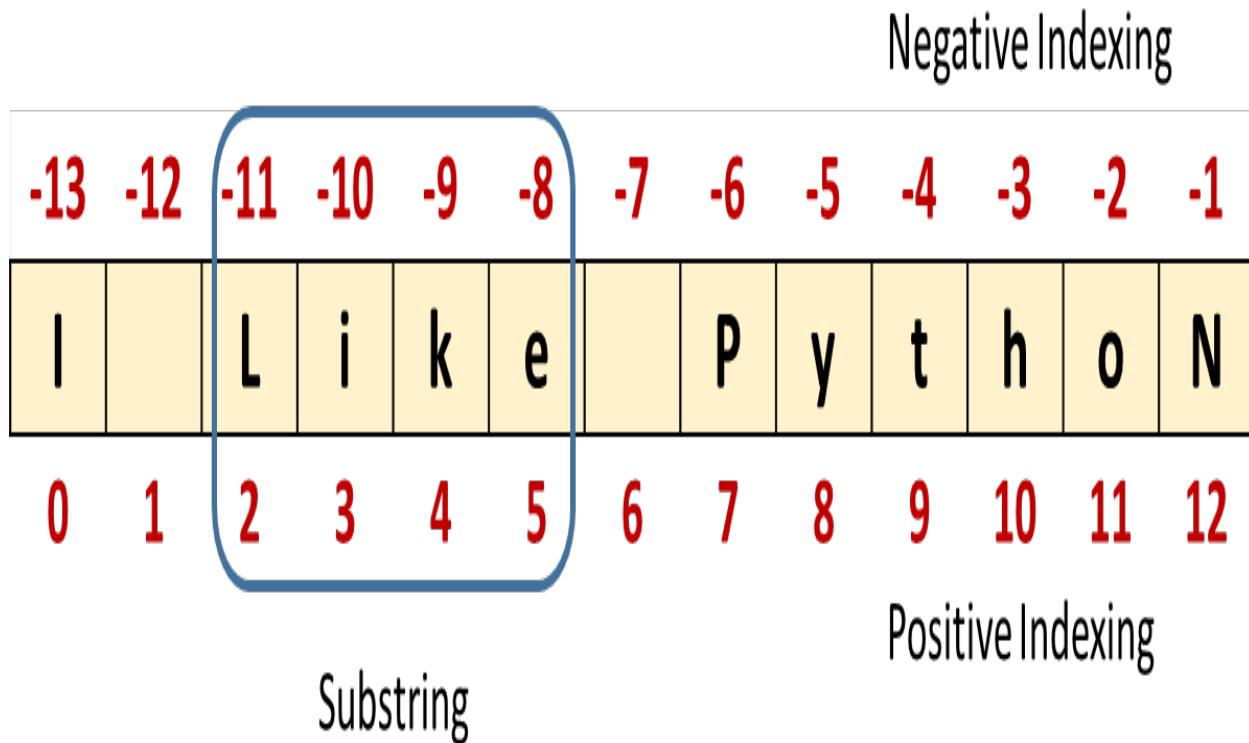
print(st[2])      #print 'L' using positive index
print(st[-11])   #print 'L' using negative index

print(st[7])      #print 'P' using positive index
print(st[-6])    #print 'P' using negative index
```



# String Slicing

- String slicing is the way of selection of substring.
- String 'Like' is substring in given 'I Like Python' string.
- [] we can access string characters by giving indexes if we use colon inside square bracket like [:] it becomes a slicing operator in Python.



# Syntax of String Slicing

- Start index: Index from which slicing starts.
- End Index: Index up to which slicing end.
- Step value is optional.

Index from which slicing start

Index up to which slicing end

**st [ start index :End Index :Step]**

String Object / Variable

- Step is Optional
- Default step value → +1
- Step value can be positive or negative

# Examples of String Slicing

Example	Explanation	Syntax
<b>st[ 2 : 6 ]</b>	It starts with the 2 <sup>nd</sup> index and ending with (6-1=5)th index.	<pre>st = 'I Like Python' print(st[2:6])</pre> <p>Like</p>
<b>st[ 0 : 6 : 2 ]</b>	It starts with index 0 <sup>th</sup> and end with (6-1=5) <sup>th</sup> . Step is 2 So, it will give value at index 0,2,4	<pre>st = 'I Like Python' print(st[0:6:2])</pre> <p>ILk</p>
<b>st[ 12 : 6 : -1 ]</b>	It starts slicing from 12 <sup>th</sup> index up to 6-1=5 <sup>th</sup> index. 12 <sup>th</sup> to 6 <sup>th</sup> in opposite direction because step is negative.	<pre>st = 'I Like Python' print(st[12:6:-1])</pre> <p>nohtyP</p>

# Contd..

Example	Explanation	Syntax
<b>st[ -11 : -7 ]</b>	It starts with -11 <sup>th</sup> index and ending with (-7-1 = -8)th index.	<pre>st = 'I Like Python' print(st[-11:-7])</pre> <p>Like</p>
<b>[ : ]</b>	The operator gives the complete original string.*	<pre>st = 'I Like Python' print(st[:])</pre> <p>I Like Python</p>
<b>[ : 6 ]</b>	Start index is missing and step is +1, So it will start from 0 till 6-1=5 <sup>th</sup> index.*	<pre>st = 'I Like Python' print(st[:6])</pre> <p>I Like</p>

# Contd..

Example	Explanation	Syntax
<code>st[ 7 : ]</code>	It will start from index 7 to last index ( Last index missing )*	<pre>st = 'I Like Python' print(st[7:])</pre> <p>Python</p>
<code>st[ :: -1 ]</code>	Step is -1, So start index will be the last index and end index will be first index when start and end is missing.	<pre>st = 'I Like Python' print(st[::-1])</pre> <p>nohtyP ekil I</p>
<code>st[ 2 : 6 : -1 ]</code>	This is explaining in Note Section ** Output – Empty string	<pre>st = 'I Like Python' print(st[2:6:-1])</pre>

# Important Points to Remember

## Note about \*, \*\*

- \* When start index is missing it will start from either first character or from last. It depends on step sign (positive or negative).
- \* When end index is missing it will execute till last character or first character. It depends on step sign (positive or negative).
- \* When we take negative steps it will scan from start to end in opposite direction.
- \*\* In the case of step is positive or negative the slicing will be done as below given algorithm.

# Comparison

When Step is Positive	When Step is negative
<pre>st [ start : end : +1 ]</pre> <p><b>Working of slicing</b></p> <p>i = start</p> <p>while i &lt; end:</p> <p style="padding-left: 40px;">//do scanning</p> <p>i = i +1</p>	<pre>st [ start : end : -1 ]</pre> <p><b>Working of slicing</b></p> <p>i = start</p> <p>while i &gt; end:</p> <p style="padding-left: 40px;">//do scanning</p> <p>i = i -1</p>
<p><b>Example –</b></p> <pre>st[6:2:1]</pre> <p>i = 6</p> <p>while i &lt; 2:</p> <p style="padding-left: 40px;">//do scanning</p> <p>i = i +1</p> <p><b>Expression i&lt;2 will evaluate to False and loop will terminate and empty string will be slice.</b></p>	<p><b>Example –</b></p> <pre>st[2:6:-1]</pre> <p>i = 2</p> <p>while i &gt; 6:</p> <p style="padding-left: 40px;">//do scanning</p> <p>i = i -1</p> <p><b>i&gt;6 will evaluate to False and loop will terminate and empty string will be slice.</b></p>

# Updation in String

- ❑ Strings in python are Immutable(un-changeable) sequences, which means it does not support new element assignment using indexes.
- ❑ Lets try to understand this concept with the help of an example

```
st = 'I Like Python'  
st[0] = 'i'
```

```
-----  
TypeError                                     Traceback (most recent call last)  
<ipython-input-10-ee7a862518bd> in <module>  
      1 st = 'I Like Python'  
----> 2 st[0] = 'i'  
  
TypeError: 'str' object does not support item assignment
```

In the above example string does not allow assigning a new element, because item assignment does not support by string.

## Contd..

1. What will be the output of the following code snippets?

```
message="welcome to Mysore"
word=message[-7:]
if(word=="mysore"):
    print("got it")
else:
    message=message[3:14]
    print(message)
```

- A. come to Myso
- B. come to Mys
- C. Icome to Mys\
- D. Icome to Myso



## Contd..

2. What will be the output of the following code Comparison?

```
print("ABES" > "AKTU")
print("AKTU" < "ABES")
```

A. True  
False

B. False  
False



## Contd..

3. What will be the output of the following code snippet?

```
example = "ABES Engineering College"  
print("%s" % example[2:7])
```

- A. ES
- B. En
- C. ES En
- D. ESEn



## Contd..

4. What will be the output of the following code snippet?

```
s1 = 'hellohow'  
for i in range(len(s1)):  
    print(s1,end="")  
    s1 = 'a'
```

- A. hellohowa
- B. hellohowaaaaaa
- C. hellohowaaaaaaaa
- D. Error



# Session Plan - Day 2

## 3.1 String

- Built in Methods
- Basic Operations
- String Formatters
- Loops with Strings
- Review Questions
- Practice Exercises

# Deletion

- In String, we can't reassign the string characters but we can delete the complete string using **del command**.
- Lets try to understand this concept with the help of an example

```
st = 'I Like Python'
print(st)
del st
print(st)

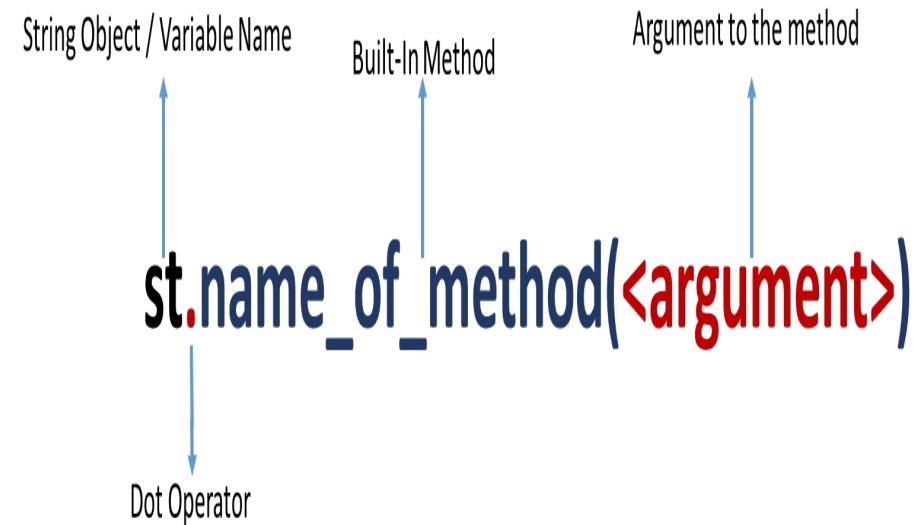
I Like Python
-----
NameError: name 'st' is not defined
Traceback (most recent call last)
<ipython-input-11-933f7d08d696> in <module>
    2 print(st)
    3 del st
----> 4 print(st)

NameError: name 'st' is not defined
```

In the above example , after deletion when we try to print the string st it gives NameError : st not defined.

# String Built in methods:

- String supports a variety of Built-In methods for achieving different types of functionalities.
- All built-in methods provide us to use as plug and play, we do not have to implement it.
- In python methods are called using dot(.) operator



# String Built in methods:

Method Name	Explanation	Code Snippet
capitalize()	Return a string with first letter capital e.g. i like python → I Like Python	<pre>st = 'i like python' stnew = st.capitalize() print(stnew)</pre> <p>I like python</p>
casefold()	Return string in lowercase	<pre>st = 'i Like python' stnew = st.casefold() print(stnew)</pre> <p>i like python</p>
count()	Return the number of occurrence of substring. In the first example “i” occurred 3 times. In the second example “like” occurred 2 times.	<pre>st = 'i like what you like' n = st.count('i') print(n)</pre> <p>3</p> <pre>st = 'I like what you like' n = st.count('like') print(n)</pre> <p>2</p>

# String Built in methods:

Method Name	Explanation	Code Snippet
endswith()	Return <b>True</b> if the string ends with the given substring otherwise return <b>False</b> .	<pre>st = 'abes.ac.in' print(st.endswith('in'))</pre> <p>True</p>
startswith()	Returns <b>True</b> if the string starts with the given substring otherwise return <b>False</b> .	<pre>st = 'abes.ac.in' print(st.startswith('abes'))</pre> <p>True</p> <pre>st = 'abes.ac.in' print(st.startswith('abesec'))</pre> <p>False</p>
find()	Return the lowest index in String where substring sub is found. In example – there are two “like”. 1 <sup>st</sup> “like” is at index-2 and 2 <sup>nd</sup> at index-16. It return 2.	<pre>st = 'I like what you like' n = st.find('like') print(n)</pre> <p>2</p>

# String Built in methods:

Method Name	Explanation	Code Snippet
lower()	Convert and return a string into lower case	<pre>st = 'I Like Python' stnew = st.lower() print(stnew)</pre> <p>i like python</p>
upper()	Convert and return a string into upper case	<pre>st = 'I Like Python' stnew = st.upper() print(stnew)</pre> <p>I LIKE PYTHON</p>
swapcase()	In swaps cases, the lower case becomes the upper case and vice versa	<pre>st = 'I Like Python' stnew = st.swapcase() print(stnew)</pre> <p>i lIKE pYTHON</p>

# String Built in methods:

Method Name	Explanation	Code Snippet
title()	Converts the first character of each word to upper case.	<pre>st = 'i like python' stnew = st.title() print(stnew)</pre> <p>I Like Python</p>
replace()	Returns a string after replacing old substring with new substring.	<pre>st = 'I like python' stnew = st.replace('python', 'java') print(stnew)</pre> <p>I like java</p>
split()	<p>Splits the string from given separator and returns a list of substring.</p> <p>Note –</p> <p>By default, value of separator is a whitespace</p>	<pre>st = 'I Like Python' stnew = st.split() print(stnew)</pre> <p>['I', 'Like', 'Python']</p> <pre>st = 'Sunday,Monday,Tuesday' stnew = st.split(',') print(stnew)</pre> <p>['Sunday', 'Monday', 'Tuesday']</p>

# String Built in methods:

Method Name	Explanation	Code Snippet
join()	Concatenate any number of strings. The string whose method is called is inserted in between each given string. The result is returned as a new string	<pre>st = 'Python' stnew = '-'.join(st) print(stnew)</pre> <p>P-y-t-h-o-n</p>
strip()	Return a copy of the string with leading and trailing whitespace removed.	<pre>st = ' I Like Python ' stnew = st.strip() print(stnew)</pre> <p>I Like Python</p>
isalnum()	Return true if the string is alphanumeric (Combination of a-z, A-Z, 0-9) Note – It will return false only if string contains special character.	<pre>st = 'Python3' print(st.isalnum())</pre> <p>True</p> <pre>st = 'Python-3' print(st.isalnum())</pre> <p>False</p>

# String Built in methods:

Method Name	Explanation	Code Snippet
isalpha()	Return true if the string contains only alphabets(Combination of a-z, A-Z)	<pre>st = 'Python' print(st.isalpha())</pre> <p>True</p>
isdecimal()	Returns True if all characters in the string are decimals	<pre>st = '123' print(st.isdecimal())</pre> <p>True</p> <pre>st = 'Python34' print(st.isdecimal())</pre> <p>False</p>
islower()	Returns True if all characters in the string are lower case	<pre>st = 'python' print(st.islower())</pre> <p>True</p> <pre>st = 'Python' print(st.islower())</pre> <p>False</p>

# String Built in methods:

Method Name	Explanation	Code Snippet
len()	Return length(No of character) of a given string. This is a generic function.	<pre>st = 'Python' print(len(st))</pre> <p>6</p>

# Basic Operations in String:

- Strings in Python support basic operations like **concatenation, replication and membership**
  - **Concatenation** means joining/combining two string into one.
  - **Replication** means repeating same string multiple times.
  - **Membership** tells a given string is member of another string or not.

# Operations in String:

Method Name	Explanation	Code Snippet
+	<p>Concatenation</p> <p>It will merge/join second string at the end of first string and return.</p>	<pre>st1 = 'Go' st2 = 'ing' print(st1+st2)</pre> <p>Going</p>
*	<p>Multiply (Replicas)</p> <p>It will repeat same string multiple times and return.</p> <p>Note – multiply string only with integer number.</p>	<pre>st = 'Python' print(st*3)</pre> <p>PythonPythonPython</p>
in	<p>Membership</p> <p>It will check a given substring is present in another string or not.</p> <p>Note – gives bool value (True/False)</p>	<pre>st = 'I Like Python' print('Python' in st)</pre> <p>True</p> <pre>st = 'I Like Python' print('Java' in st)</pre> <p>False</p>

# Operations in String:

Method Name	Explanation	Code Snippet
Not in	It's reverse of Membership	<pre>st = 'I Like Python' print('Java' not in st)</pre> <p>True</p>

# String Formatters:

- String formatting is the process of infusing things in the string dynamically and presenting the string.

## Why to use String Formatter??

- For different types of requirement to print the string in a formatted manner.
- Here, **format** implies that in what look and feel we want our strings to get printed.
- Python string provides a number of options for **formatting**.

# String Formatters:

- ❑ String formatting is the process of infusing things in the string dynamically and presenting the string.

## Why to use String Formatter??

- For different types of requirement to print the string in a formatted manner.
- Here, **format** implies that in what look and feel we want our strings to get printed.
- Python string provides a number of options for **formatting**.

# String format Style:

Escape Sequence	Explanation	Example
\newline	Ignores newline	<pre>st = 'Hello "how" are you.\nI am fine' print(st)</pre> <p>Hello "how" are you.I am fine</p>
\	Backslash Write a backslash in string	<pre>st = 'https:\\\\abes.ac.in' print(st)</pre> <p>https:\\abes.ac.in</p>
'	If we need to use single quotes in our string like Good Morning! Mr. 'BOB'	<pre>st = 'Good Morning! Mr. \'BOB\'' print(st)</pre> <p>Good Morning! Mr. 'BOB'</p>

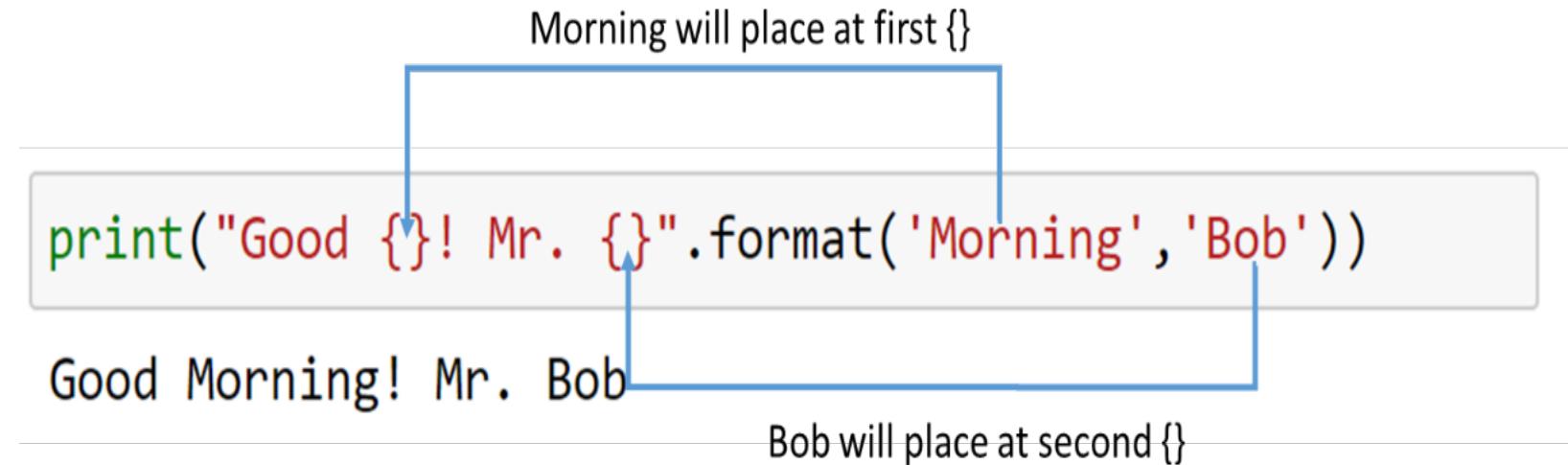
# String format Style

Escape Sequence	Explanation	Example
\"	If we need to use double quotes in our string like Good Morning! Mr. 'BOB'	<pre>st = "Good Morning! Mr. \"BOB\" " print(st)</pre> <p>Good Morning! Mr. "BOB"</p>
\n	Newline	<pre>st = "Good Morning!\nMr. BOB" print(st)</pre> <p>Good Morning! Mr. BOB</p>

# String Format()

Method 1: It is a beneficial method for formatting strings; it uses {} as a placeholder.

- We have placed two curly braces and arguments that give the format method filled in the same output.



# String Format()

- If we want to change the order of , we can give an index of parameters of format method starting with 0th index.

```
print("{0} and {1} play football".format('Bob', 'Ram'))
```

Bob and Ram play football

0<sup>th</sup> index

1<sup>st</sup> index

```
print("{1} and {0} play football".format('Bob', 'Ram'))
```

Ram and Bob play football

# String Format()

- If we want to change the order of , we can give an index of parameters of format method starting with 0th index.

```
print("{0} and {1} play football".format('Bob', 'Ram'))
```

Bob and Ram play football

0<sup>th</sup> index

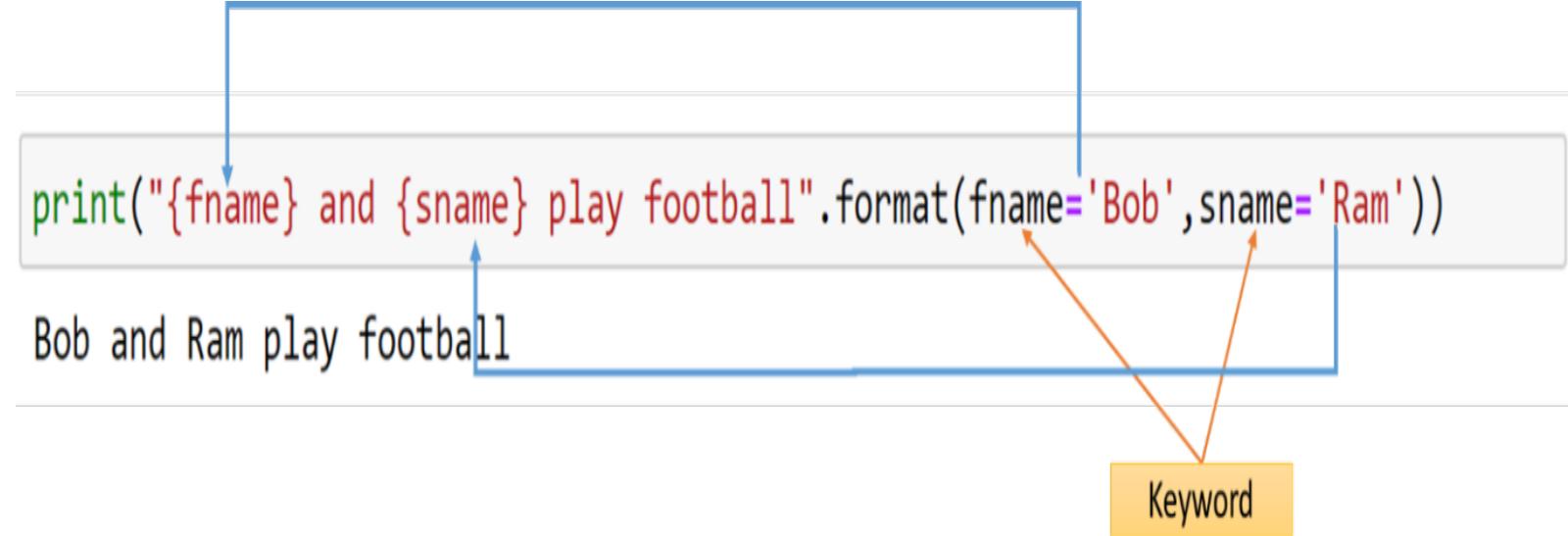
1<sup>st</sup> index

```
print("{1} and {0} play football".format('Bob', 'Ram'))
```

Ram and Bob play football

# String Format()

- We can also use keywords arguments in format method, as shown in the following example, as shown in figure.



# String Format()

- We can also use format specifier in format method like in language 'C'. Format specifier are used to do following.

- Represent value of amount = 12.68456 at two decimal place
- Represent value of integer in Binary, Octal or Hexadecimal etc.

String { Value index : conversion }.format(value)

```
val = 10
print("In Binary {0:b}".format(val))
```

In Binary 1010

Note: In the above example b is used for binary representation.

# Format Specifiers

Format Specifier	Explanation	Example
b	Use for Binary	<pre>val = 10 print("In Binary {0:b}".format(val))</pre> <p>In Binary 1010</p>
o	Use for Octal	<pre>val = 10 print("In Octal {0:o}".format(val))</pre> <p>In Octal 12</p>
X or x	Use for Hexadecimal	<pre>val = 10 print("In Hexadecimal {0:x}".format(val)) print("In Hexadecimal {0:X}".format(val))</pre> <p>In Hexadecimal a In Hexadecimal A</p>

# Format Specifiers

Format Specifiers	Explanation	Example
d	Use for Decimal	<pre>val = 10 print("In Decimal {0:d}".format(val))</pre> <p>In Decimal 10</p>
f	Use for Float Note – Place .n before f, for representing floating point precision. n is decimal places.	<pre>val = 10.8934 print("In Float {0:f}".format(val)) print("Two decimal point {0:.2f}".format(val))</pre> <p>In Float 10.893400            Two decimal point 10.89</p>

# Formatting using f-string

- ❑ Fstring is the way of formatting as **format method** does but in an easier way.
- ❑ We include 'f' or 'F' as a prefix of string.

```
a = 10
b = 20
c = a+b
print(f"Sum of {a} and {b} = {c}")
```

Sum of 10 and 20 = 30

# Can you answer these questions?

1. What will be the output of the following code snippet?

```
line = "Hello how are you"
L = line.split('a')
for i in L:
    print(i, end=' ')
```

- A. Hello how are you
- B. Hello how are
- C. hello how are you
- D. Hello how re you

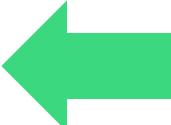


# Can you answer these questions?

2. What will be the output of the following code snippet?

```
example="ABES Engineering College"  
example[::-1].startswith("A")
```

- A. True
- B. False
- C. Error
- D. None



# Loops with Strings:

- Strings are sequence of character and iterable objects, so we can directly apply for loop through string.

## Example-1

Scan/Iterate each character of string through index using for loop.

```
st = 'Mango'  
for i in range(len(st)):  
    print(st[i])
```

M  
a  
n  
g  
o

len(st) gives → 5  
range(len(st)) → range(5) → 0,1,2,3,4

st[0] → M  
st[1] → a  
st[2] → n  
st[3] → g  
st[4] → o

## Example -2

Scan/Iterate each character of string directly using for loop

```
st = 'Mango'  
for val in st:  
    print(val)
```

M  
a  
n  
g  
o

# Loops with Strings:



downloaded from [pptkywallpapers.com](http://www.pptkywallpapers.com)

**Example 3 – Write a program to print number of alphabets and digits in a given string.**

```
st = 'NH-24, Delhi Hapur By pass Vijay Nagar 201009'
alphabet = 0
digit = 0
for val in st:
    if val.isalpha():
        alphabet = alphabet + 1
    if val.isdigit():
        digit = digit +1
print(f"Total No of alphabet = {alphabet}")
print(f"Total No of Digit = {digit}")
```

Total No of alphabet = 28

Total No of Digit = 8

## Example 4

To add 'ing' at the end of a given string (length should be at least 3).

- If the given string already ends with 'ing' then add 'ly' instead.
- If the string length of the given string is less than 3, leave it unchanged.

Sample String : 'abc'

**Expected Output : 'abcing'**

Sample String : 'string'

**Expected Output : 'stringly'**

.

```
st = input("Enter any String")
if len(st)>=3:
    if st.endswith('ing'):
        print(st+'ly')
    else:
        print(st+'ing')
else:
    print(st)
```

```
Enter any Stringabc
abcing
```

# Session Plan - Day 3

## 3.2 List

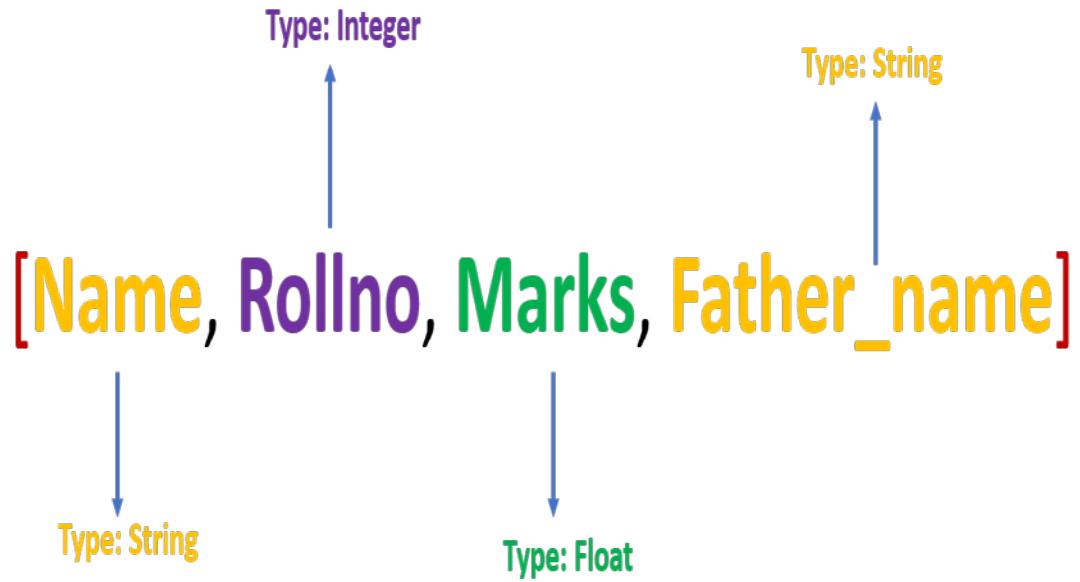
- **Creation**
- **Accessing**
- **Updation**
- **Review Questions**
- **Practice Exercises**

- Python **List** is the most commonly used sequence.
- Important points about list are as follows.
  - List elements are enclosed in **square brackets []** and are comma separated.
  - List is the sequence of class type '**list**'.
  - List can contain elements of different data types.
  - List is a mutable(changeable) sequence, would be discussed in detail in 3.2.3
  - List allows duplicate elements.
  - List elements are ordered, it means it give specific order to the elements, if new element is added, by default it comes at the end of the list.

## Real World Scenario:

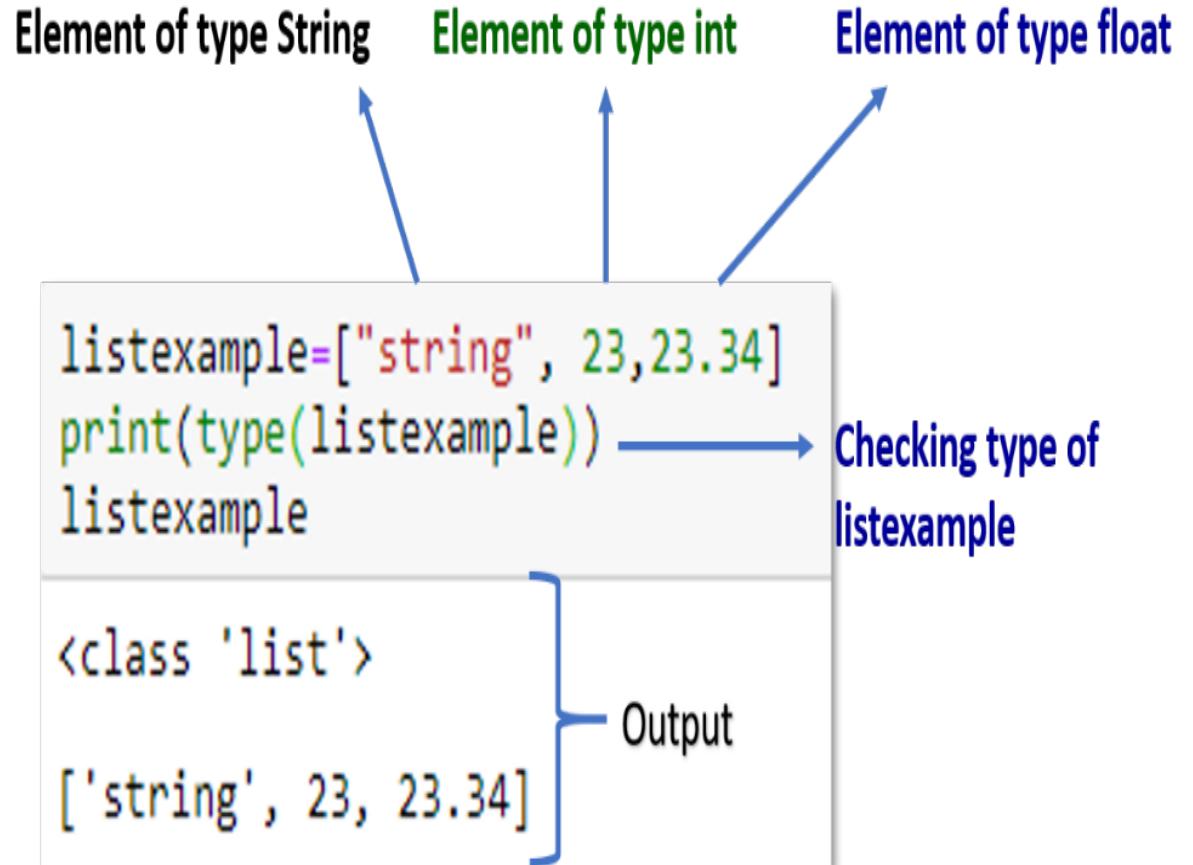
- List is used when there is a possibility of elements of different data type.
- For example record of a particular student, having name as string, roll.no as integer, marks as float, father's name as string.
- To contain this record list is the appropriate sequence.

## List Representation



# List Example

- In this example a variable named listexample has been created and three elements have been assigned.
- As we have enclosed elements in square brackets, this makes listexample is of type list.



# Ordered Property of List

- Ordered sequence of the list means the order in which elements appear is unique
- The position of every element is fixed.
- If we change the position of any element, then list would not be the same anymore.

String and 23 swapped their position in new list



```
listexample=["string", 23,23.34]  
listexample1=[23,"string",23.34]  
listexample==listexample1
```

False } Output

# Creation of List

List can be created by many ways as follows .

## Creation of Empty List

```
s=list()  
print(s)
```

[ ]→ Empty list created

```
s=[]  
print(s)
```

[ ]→ Empty list created

**Using list class**

**Using empty square brackets**

## Creation of Non Empty List

```
s=list([1,"Hello"])  
s
```

[1, 'Hello']→ list created

```
s=[1,"Hello"]
```

s

[1, 'Hello']→ list created

**Using list class**

**Using square brackets only**

## Indexing the List

- In the list index are started from 0, it means first element takes 0 index and it increases like 0,1,2...(n-1).
- List also supports negative indexing.
- It means last element can be accessed using -1 index, second last as -2 and so on.



- Slicing is used to get substring from a string.
- We use Slice operator in the list to get a sub list.

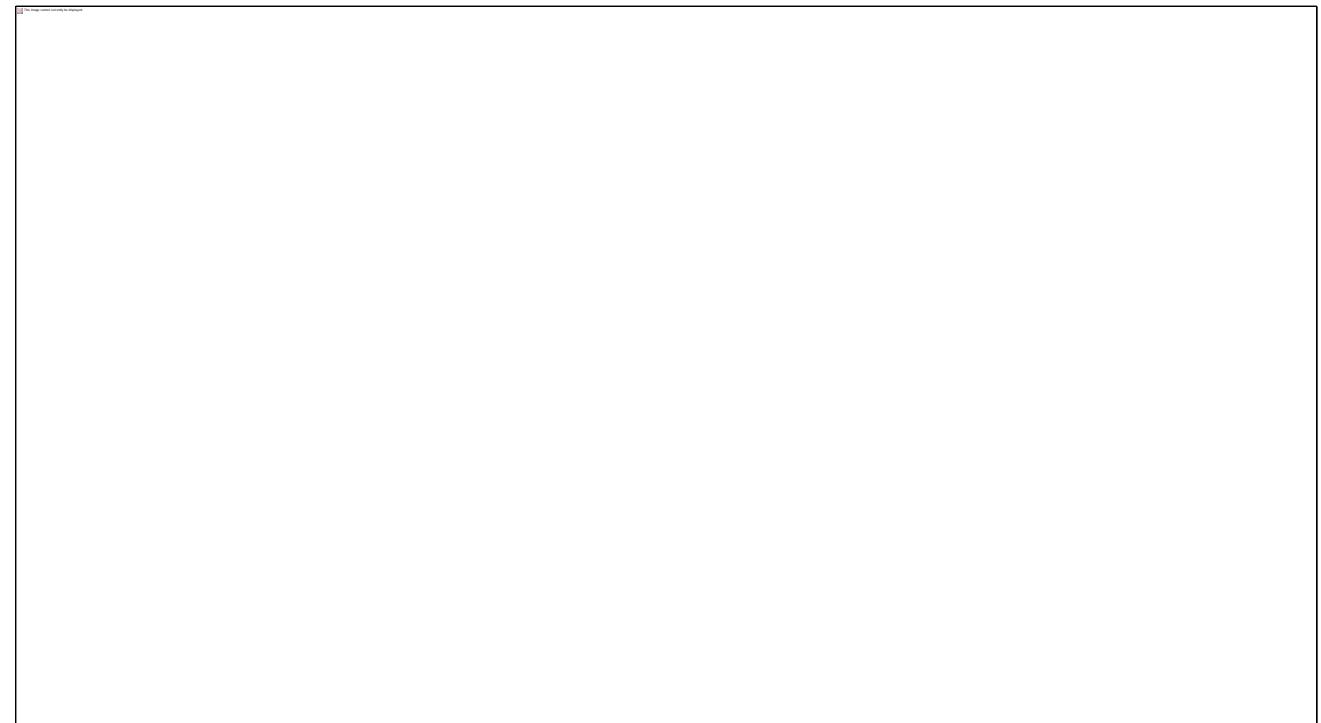
Start= Optional argument by default value is 0.

End = Optional argument by default value is number of elements in the list. If any number given, then value is taken as number-1.

Step=Optional argument by default value is 1.

Colon 1: Required

Colon 2: Optional



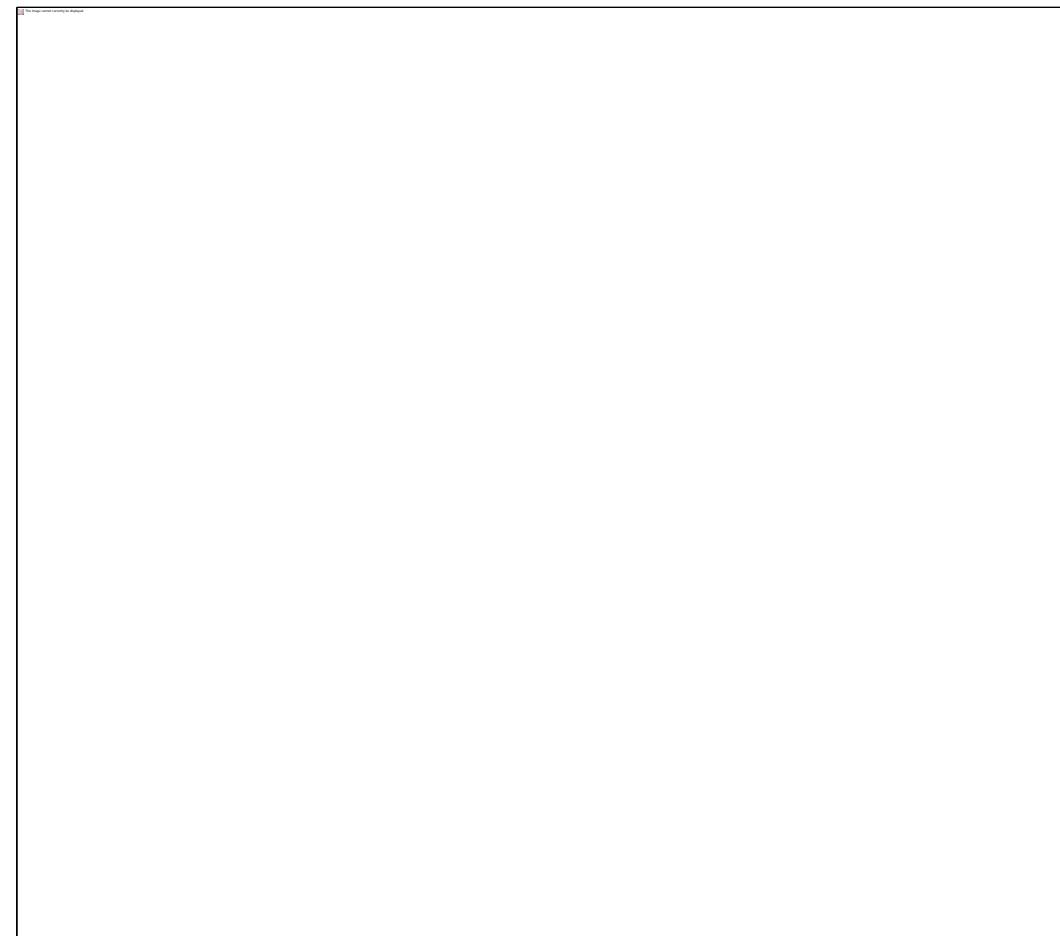
# Slicing Example



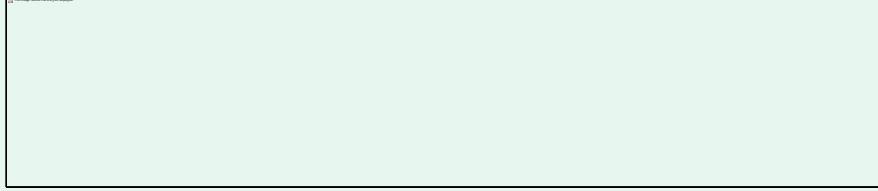
Estd. 2000

downloaded from www.wallpapers.com

- Single colon[:] - It is used to print the entire list.
- Double colon[::] - It is used to print the entire list.
- L[2::] - It is used to print the list starting from index 2 till the last element of the list.
- L[2:5:] - We have set start and end both as 2 and 5, so it's starting from 2 and ending with (5-1) and printing values for indexes 2,3 and 4 index.
- L[2:5:2] - end and step has been set as 2,5 and 2. So output substring starting from index 2, ending with 5-1=4 and step counter is 2, so its skipping alternate element.



# Examples of List Slicing

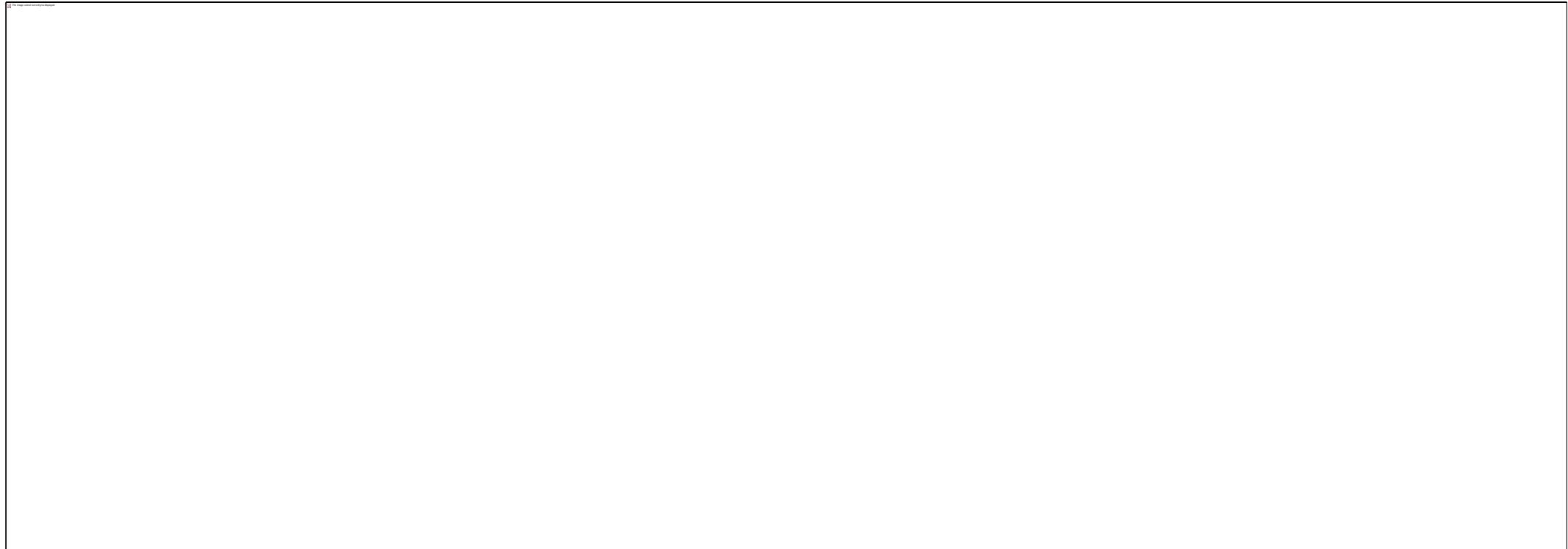
Operator	Explanation	Example
[:]	This gives a complete original list	
[1:3]	It starts with 1 <sup>th</sup> index and ending with (3-1=2) index	
[2:]	Sub list starts with index 2 and ends at last element as nothing specified on right side of the colon	

# Examples of List Slicing

Operator	Explanation	Example
<code>[1:4:]</code>	Start=1, stop=4 and by default step=1.	
<code>[1:4:2]</code>	Start=1, stop=4 and by step=2.  As the step is 2, it took every 2 <sup>nd</sup> element.	
<code>[::-1]</code>	Starting is 0th index and stopping at last index and step is negative count so that it would print in reverse order	

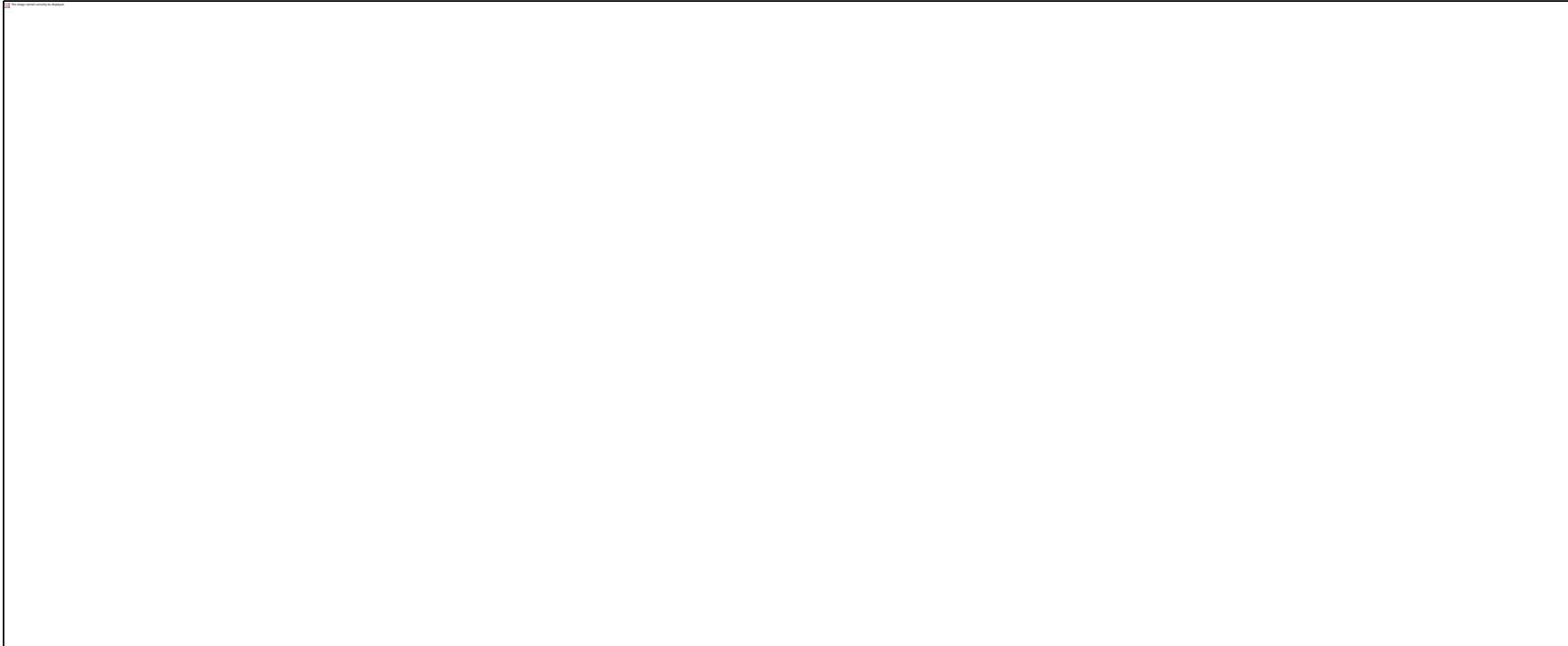
# Updation in List:

- ❑ List updating involves insertion of new elements, deletion of elements or deletion of complete list.
- ❑ List is a mutable sequence it means it allows changes in the elements of list.



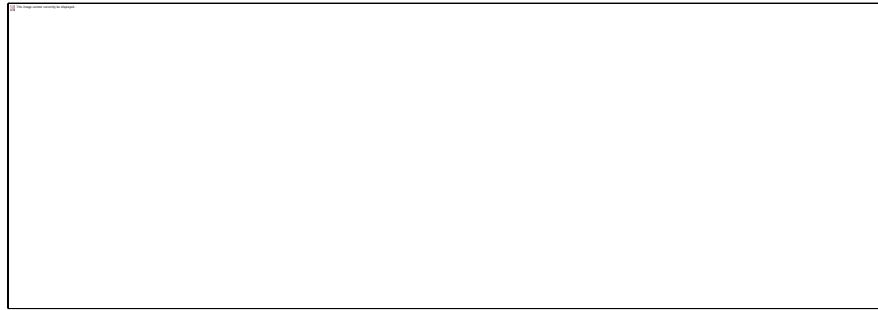
# Updation in List:

- List updating involves insertion of new elements, deletion of elements or deletion of complete list.
- List is a **mutable** sequence it means it allows changes in the elements of list.



# Can you answer these Questions

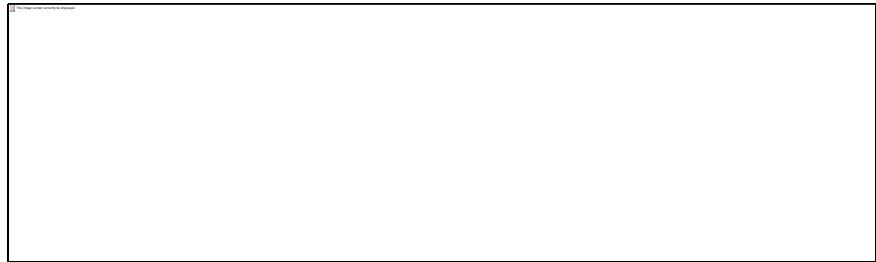
1.What is the output of the following code?



- A.[‘XY’,’YZ’] ←
- B.[‘xy’,’yz’]
- C. None of the above
- D. Both of the above

# Can you answer these Questions

2.What is the output of the following code?



- A. ['g','h','k','l','m']
- B. ['g','h','k','l','m',9]
- C. ['g','h','k','l','m',8,9] 
- D. ['g','h','k','l','m',9,8]

# Session Plan - Day 4

## 3.2 List

- **Updation in List**
- **Operations in List**
- **Built in Methods**
- **Operations**
- **Loops**
- **Nested list**
- **List Comprehension**
- **Review Questions**
- **Practice Exercises**

# Add new element in List:

- We can add elements to the existing list using append function.
- Append function always add the element at the end of the list.

Syntax:

**Listname.append (element to be added)**

In this example element 5 has been appended in the originally existed list. It's being added in the last



# Changing new element in List:

- Value of the element at index 3 has been assigned new value as 5, so element in the output list is changed from 4 to 5.



# Deletion in List:

List elements can be deleted.

- Using del command If we know the position(index) of the element which is to be deleted.
- We can use the remove method by giving the specific element as given in the example If we know the position(index) of the element.

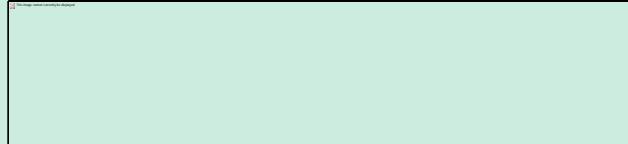
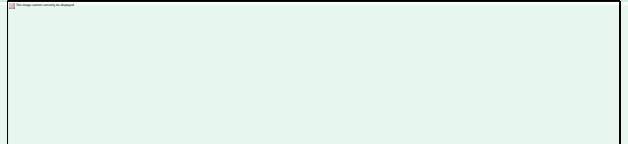


# Deletion in List:

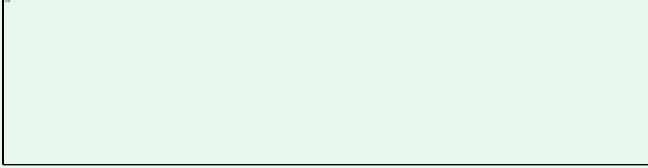
- list1 has been deleted and then we are trying to print it and its giving error because it does not exist now.



# Built in Methods in List

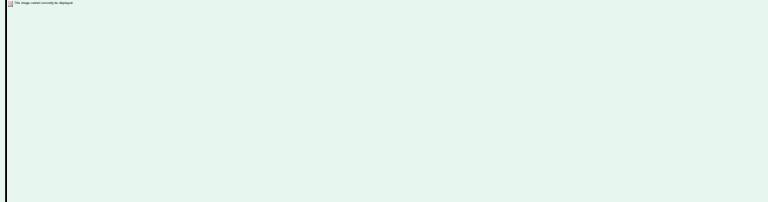
Method	Remark	Example
<code>len()</code>	<b>It calculates the length of the list or the number of elements in the list.</b>	
<code>max(list)</code>	<b>It returns the maximum element from the list</b>	
<code>min(list)</code>	<b>It returns a minimum element from the list</b>	
<code>list(seq)</code>	<b>It converts into any sequence into a list</b>	

# Contd..

Method	Remark	Example
<code>pop()</code>	<p><b>It deletes the last element from the list</b></p> <p><b>Note – We can also pass index as argument in <code>pop()</code> to delete a specific index value.</b></p>	
<code>count()</code>	<b>It counts the occurrences of a particular element in the list</b>	
<code>sort()</code>	<b>Sort the elements of the list in ascending order</b>	
<code>reverse()</code>	<b>Sort the elements of the list in reverse order</b>	

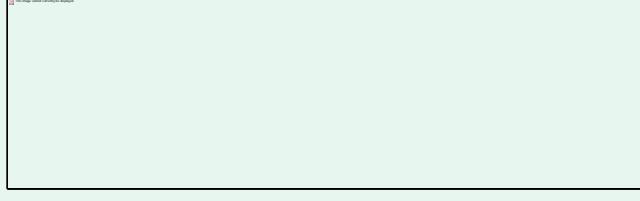
# Operations on List

- Python supports variety of operations on the list

Operation	Remark	Example
<b>Concatenation</b>	Operation adds two list elements	
<b>Repetition</b>	It repeats the list specified number of times	

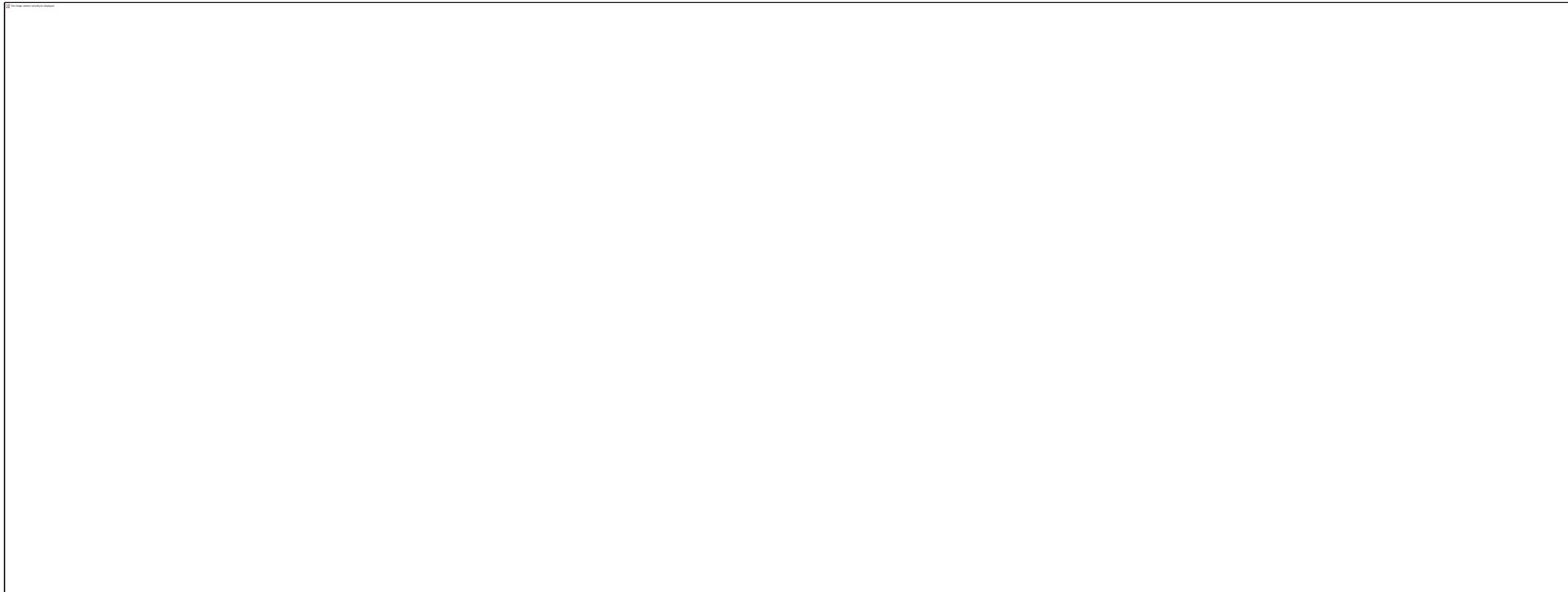
# Contd..

- Python supports variety of operations on the list

Operation	Remark	Example
<b>Membership</b>	To check whether an element belongs to the list or not	
<b>Membership not</b>	I return true if an element that does not belong to the list	

# Example

- Write Python Program to swap elements in the list.



# Practice Problems

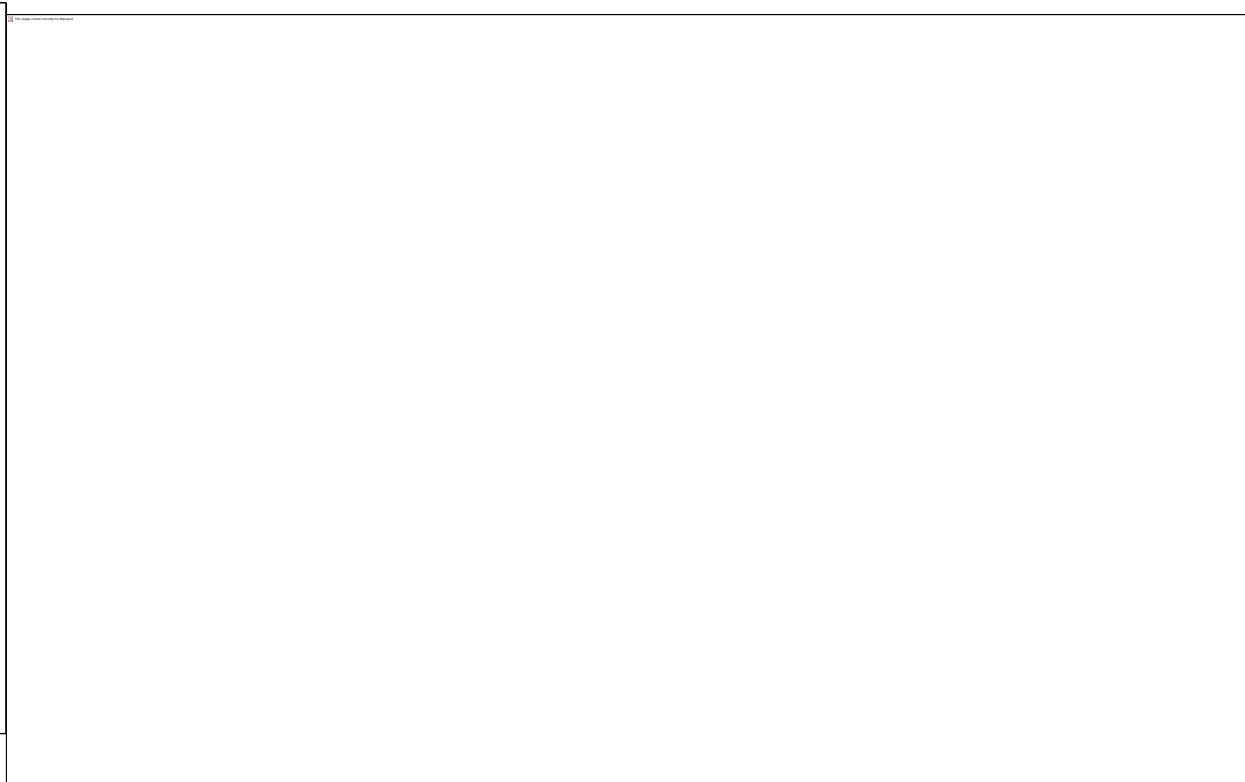
- Take a list input from user having integer elements and calculate sum and average of the list.
- Take an input list and swap string elements of the list with empty string.

# Loops with List

□ While loop with list



□ For Loop with list



# Example

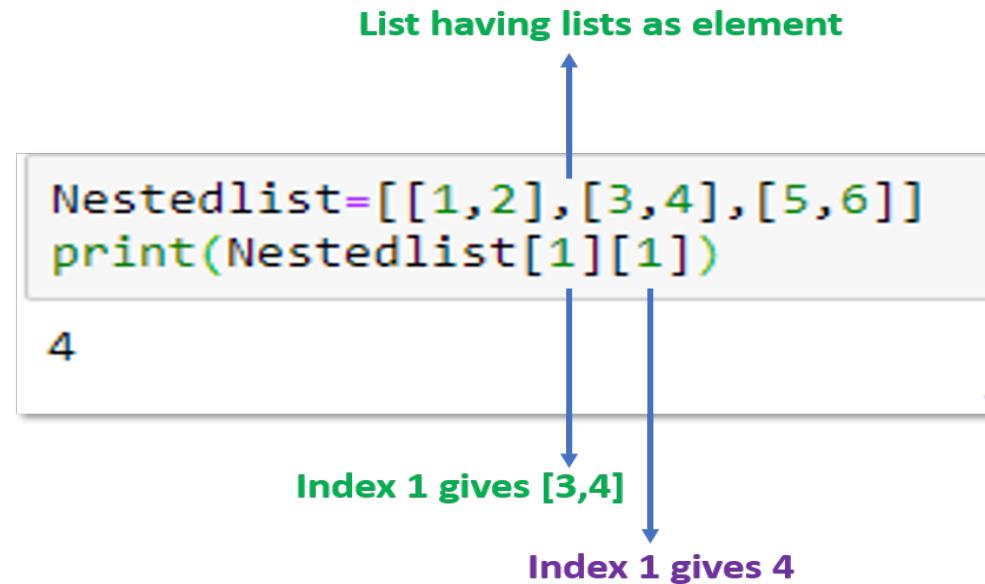
- Write a python program to print all positive number of a list.

Practice Exercise:

- Write a python program to remove duplicate from the list.
- Write a python program to count positive, negative and string type elements.

# Nested List

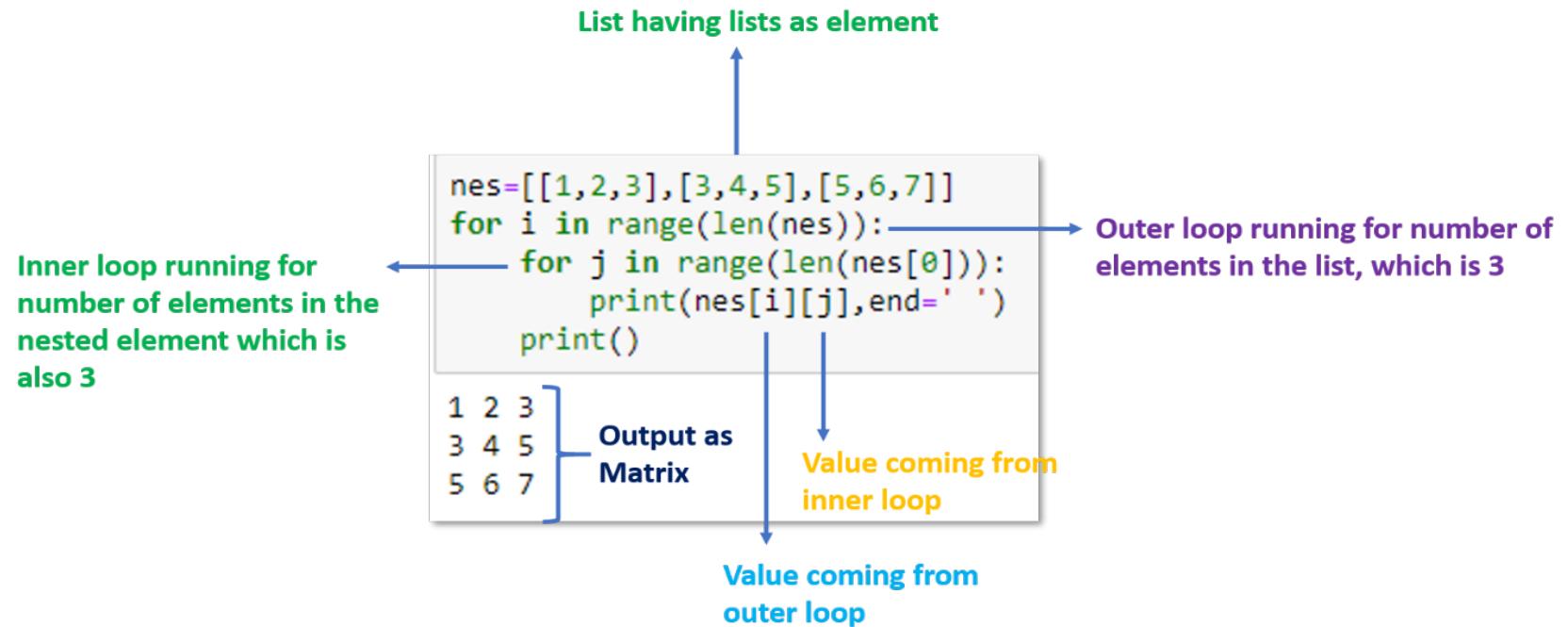
- When we have an element of a list in the form of the list itself, it is called Nested List.
- Elements of the nested list can be accessed using the 2-D indexing access method.



In this example, first [1] denotes [3,4] and second [1] represents 4, so it gives output as 4.

# Nested List as a Matrix

- We can represent nested list as matrix also.
- For this we would use nested for loop.
- Outer loop would run for number of elements in the list and inner loop would consider individual element of the nested list as a list.



# Example

- Create a flat list from a nested list.

```
nestedlist=[[1,2],[3,4],[5,6,7]] } Given list
newlist=[] → Empty list created for output
for i in range(len(nestedlist)):
    for j in range(len(nestedlist[i])):
        newlist.append(nestedlist[i][j]) } Nested for loop to access
                                         nested list elements
print(newlist) → Printing of new list

[1, 2, 3, 4, 5, 6, 7] } Output
```

## Practice Exercise:

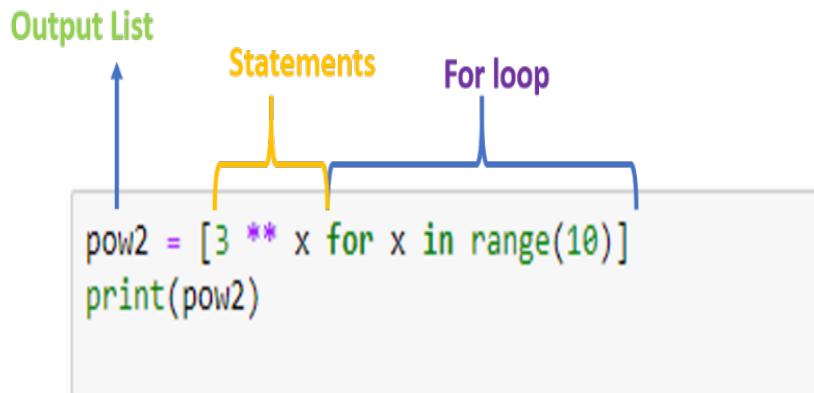
- Using nested list print transpose, of a given matrix
- Print reverse order of a nested list

# List Comprehensions

- Let's suppose we want to write a program to calculate powers of 3 for a given range.
- Traditional Programming
- Python provide us writing iterative programs in much lesser lines called List comprehension.
- The same problem has been solved using list comprehension.

```
pow2 = []
for x in range(10):
    pow2.append( 3** x)
print(pow2)
```

[1, 3, 9, 27, 81, 243, 729, 2187, 6561, 19683]



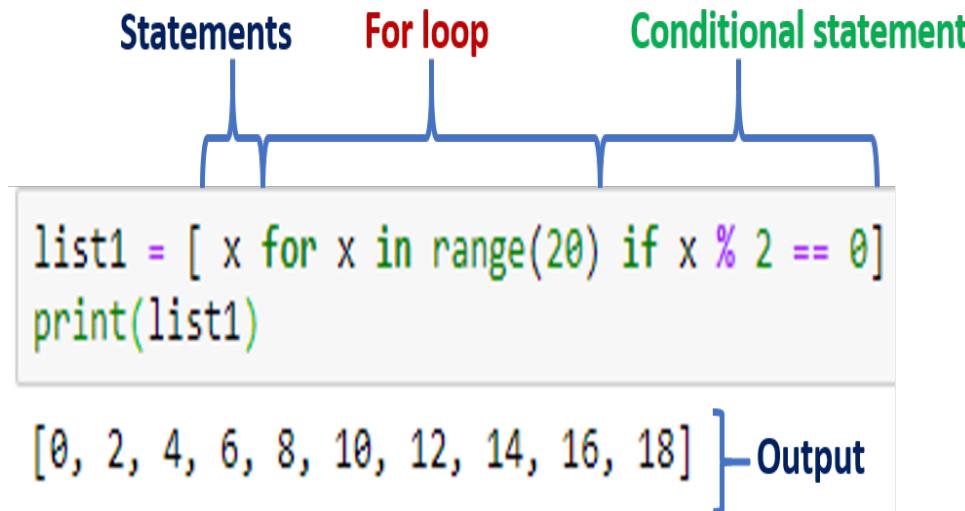
[1, 3, 9, 27, 81, 243, 729, 2187, 6561, 19683] } Output

# List Comprehension

## Syntax:

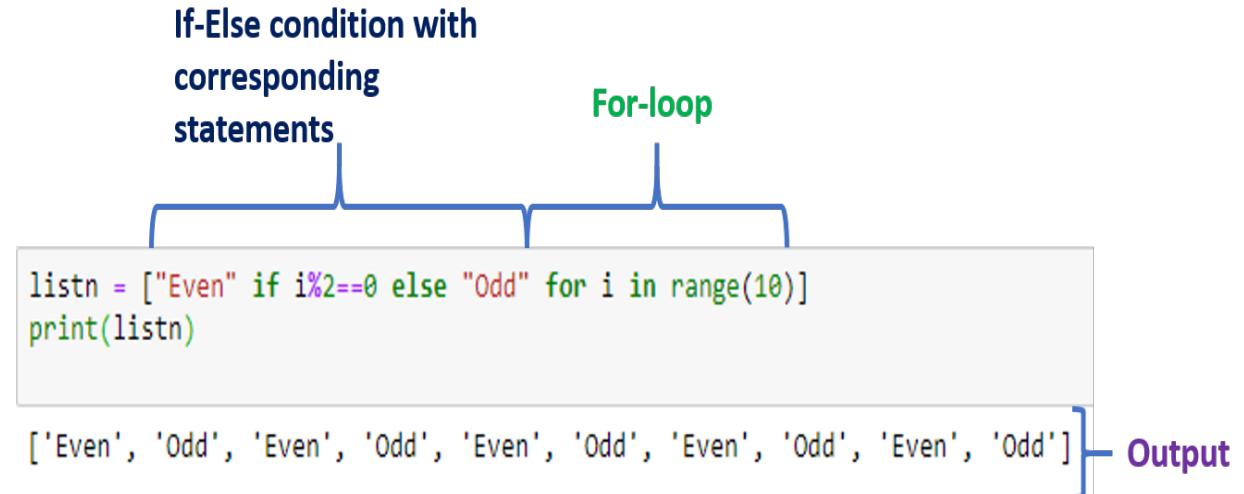
[ *statements* for an item in list ]

- We have the facility of adding conditionals in the list comprehension.



## IF ELSE :

- We should notice one change that because in if-else case output is separated based on the condition.
- if-else and conditions has been written along with statements.



# Can you answer these questions?

1. What will be the Output of the following code?

```
t = [p**+1 for p in range(6)]
```

- A.[0,1,2,3,4,5] 
- B.[0,1,2,3,4,5,6]
- C.[0,1,2,3]
- D. None of the Above

# Can you answer these questions?

2. What will be the Output of the following code?

```
B=[[3, 4, 5],[6,7,8],[9,10, 11]]  
[B[i][len (B)-1-i] for i in range (len (B))]
```

A. [5,7,9]



B.[3,4,5]

C.[6,7,8]

D.[9,10,11]

# Session Plan - Day 5

## 3.3 Tuple

- **Creation**
- **Assessing**
- **Modification/Updation**
- **Built in Methods**
- **Operations**
- **Review Questions**
- **Practice Exercises**

# Session Plan - Day 5

## 3.3 Tuple

- **Creation**
- **Assessing**
- **Modification/Updation**
- **Built in Methods**
- **Operations**
- **Review Questions**
- **Practice Exercises**

# Tuple

- ❑ Tuple is an immutable (unchangeable) ordered collection of elements of different data types.
- ❑ Generally, when we have data to process, it is not required to modify the data values.
- ❑ Take an example of week days, here the days are fixed. So, it is better to store the values in the data collection, where modification is not required.

Square brackets are the representation of list

```
list_days = ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri']  
tuple_days = ('Mon', 'Tues', 'Wed', 'Thurs', 'Fri')
```

Round brackets are the representation of tuple

# Creation of Tuple

## Creation of Empty Tuple

```
Variable name      Tuple Symbol
↑                 ↑
new_tuple = ()
```

- ❑ In above example, the () round brackets are used to create the variable of tuple type.
- ❑ We can also call the class to create a tuple object as shown in example below.

```
Variable/object name      Class <tuple>
↑                         ↑
new_tuple1 = tuple()
```

## Creation of Non Empty Tuple

### ❑ Syntax of Creating Non Empty Tuple

```
Integer String Boolean
↑     ↑     ↑
employee=(1, 'Steve', True, 25, 12000) # heterogeneous data tuple
print(employee)
(1, 'Steve', True, 25, 12000)
```

} Output

# Example

- Write a program in python to create one element in tuple

```
a = (1)           Trying to create tuple with single integer value
b = ('ABES')      Trying to create tuple with single string value
c = ('college',) 

print(type(a))
print(type(b))
print(type(c))

<class 'int'>
<class 'str'>
<class 'tuple'>           To make a single element tuple, add comma after a value
```

- If you want to create a tuple with single value, it is required to add a comma after the single value as shown in the above example **c=('college',)**.
- If the comma is not placed, then the single value **a= (1)** or **b=(ABES)** will be treated as an independent value instead of the data collection.

# Packing and Unpacking of Tuple

- ❑ Tuple can also be created without using parenthesis; it is known as **packing**.
- ❑ Write a program to create tuple without parenthesis.

```
newtup4=3,4,5,"hello"
print(type(newtup4))
print(newtup4)
```

```
<class 'tuple'>
(3, 4, 5, 'hello')
```

In above example 1, **newtup4=3,4,5," hello"** creates a new tuple without parenthesis.

# Unpacking of Tuple

- ❑ Write a program to unpack elements in tuple
- ❑ Unpacking is called when the multiple variables is assigned to a tuple; then these variables take corresponding values.

```
newtuple=3,4,5,"hello" #packing
print(newtuple)
a,b,c,d=newtuple      #unpacking
print(a,b,c,d)
```

```
(3, 4, 5, 'hello')
3 4 5 hello
```

## Practice Questions:

- ❑ Write a Python program to create the colon of a tuple.
- ❑ Write a Python program to unpack a tuple in several variables.

# Accessing Elements in Tuple

- ❑ Tuple elements can be accessed using indexes, just like String and List .
- ❑ Each element in the tuple is accessed by the index in the square brackets [].
- ❑ An index starts with zero and ends with (number of elements - 1)

Example:

Write a program to access index 1 element from tuple.

```
newtup=("hello",2,3,23.45)
print(newtup[1]) ← Accessing the index one of tuple
```

2 ← Output

In above example 1: tuple **newtup** carries multiple types of data in it as “hello” is string, 2 is integer, and 23.45 is float. If we can fetch the index 1 element using **print(newtup[1])**.

## Contd...

- Write a program to access index 0 element from tuple.

```
tuple_days = ('Mon', 'Tues', 'Wed', 'Thurs', 'Fri')  
print(tuple_days[-1]) ← Accessing the index -1 of tuple  
Fri ← Output
```

- Basically, the -1 index will return the last value of tuple.
- Indexes start from zero, and it goes up to a number of elements or say -1.
- Take another example to fetch the -1 index from tuple.
- last element is accessed by **print(tuple\_days[-1])**.

# Indexing in Tuple and Slicing

- ❑ Slicing in a tuple is like a list that extracts a part of the tuple from the original tuple.
- ❑ Write a program to access elements from 0 to 4 index in tuple.

```
tuple_days = ('Mon', 'Tues', 'Wed', 'Thurs', 'Fri')  
print(tuple_days[0:4])      ← Accessing multiple indexes of tuple  
('Mon', 'Tues', 'Wed', 'Thurs') ← Output
```

- ❑ Write a program to access elements from 1 to 3 index in tuple.

```
newtup=("hello",2,3,23.45)  
print(newtup[1:3])  
(2, 3)
```

# Modification/Updating a Tuple

- If we want to change any of the index value in tuple, this is not allowed and throw an error.
- Tuple object does not support item assignment.
- Here, we are taking the same above example of days and showing the immutable characteristic of tuple

We want to change the value 'Mon' to 'Monday'

```
tuple_days = ('Mon', 'Tues', 'Wed', 'Thurs', 'Fri')
tuple_days[0] = 'Monday'

-----
TypeError                                     Traceback (most recent call last)
<ipython-input-2-9c932d3b637c> in <module>
      1 tuple_days = ('Mon', 'Tues', 'Wed', 'Thurs', 'Fri')
      2
----> 3 tuple_days[0] = 'Monday'

TypeError: 'tuple' object does not support item assignment
```

# Practice Exercises

- Write a Python program to get the 4th element and 4th element from last of a tuple.
  
- Write a Python program to check whether an element exists within a tuple.

# Built in Methods in Tuple

Tuple has 2 Built in methods

Method name	Remark	Example
<b>count()</b>	It returns the occurrences of the value given	<pre>newtup=("hello",2,3,23.45,2,3) print(newtup.count(2))</pre> <p>2</p>
<b>index()</b>	It returns the index of the specified value	<pre>newtup=("hello",2,3,23.45) print(newtup.index(2))</pre> <p>1</p>

```
tup1=(2,3,4)
3 not in tup1
False
4)
```



Estd. 2000

downloaded from pickywallpapers.com

# Operations on Tuple

Operations	Remarks	Example
Concatenation	Add two tuples	<pre>tup1=(2,3,4) tup2=(5,6,7) tup1+tup2</pre> (2, 3, 4, 5, 6, 7)
Multiplication	Creates copies of the tuple	<pre>tup1=(2,3,4) tup1*2</pre> (2, 3, 4, 2, 3, 4)
Membership	To check whether an element belong to tuple or not	<pre>tup1=(2,3,4) 3 in tup1</pre> True
Not Membership	Would return true if element does not belong to tuple.	<pre>tup1=(2,3,4) 3 not in tup1</pre> False

# Loops and Conditions on Tuples

- ❑ Tuple can be traversed using Loop
- ❑ Take the tuple named *tuple\_days* and print all its elements

```
tuple_days = ('Mon', 'Tues', 'Wed', 'Thurs', 'Fri')

for i in tuple_days:
    print(i)
```

Mon  
Tues  
Wed  
Thurs  
Fri

} Output

- ❑ **In this example** The elements are printed while iterating through for loop condition **for in tuple\_days** and then we print the values of **i** in each iteration.

# Loops and Conditions on Tuples

Example:

- Let have a look to the example where we are placing a condition to return the values having word length greater than 3.

```
tuple_days = ('Mon', 'Tues', 'Wed', 'Thurs', 'Fri')

for i in tuple_days:
    if len(i)>3:
        print(i)
```

Tues }  
Thurs      Output

Condition in Iteration on tuple

- In above example , The elements are printed while iterating through for loop condition **for in tuple\_days** and then we print the values of **i** in each iteration with the condition **if(len(i)>3)**

# Comparison between List and Tuple

List	Tuple
Mutable Sequence	Immutable Sequence
Accession Slower than Tuple because of mutable property	Faster access of elements due to immutable property
It cannot be used as Dictionary keys	Can work as a key of the dictionary
Not suitable for application which needs write protection	It suits such scenarios where write protection is needed.

# Practice Exercises:

**Exercise:** Write Python program to join two input tuples, if their first element is common.

```
tup1=tuple(input("enter first tuple").split())
tup2=tuple(input("enter second tuple").split())

if(tup1[0]==tup2[0]):
    print(tup1+tup2)
else:
    print("First element not same")
```

```
enter first tuple2 4 5 6
enter second tuple2 3
('2', '4', '5', '6', '2', '3')
```

---

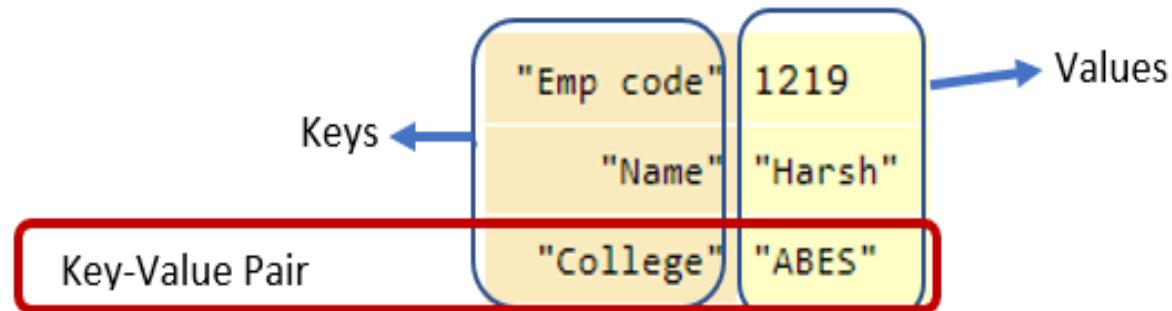
# Session Plan - Day 6

## 3.4 Dictionaries

- Creation
- Assessing
- Modification/Updation
- Nested Dictionary
- Built in Methods
- Review Questions
- Practice Exercises

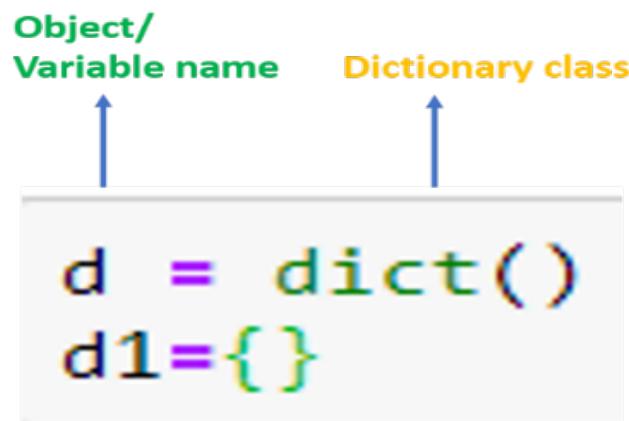
# Dictionaries

- Dictionary is a unique data collection of Python which stores the key-value pair.
  - The user can add an element by giving a user-defined index called a **key**.
  - Each key and its corresponding value makes the key-value pair in dictionary.
  - This key-value pair is considered as one item in dictionary.



# Introduction

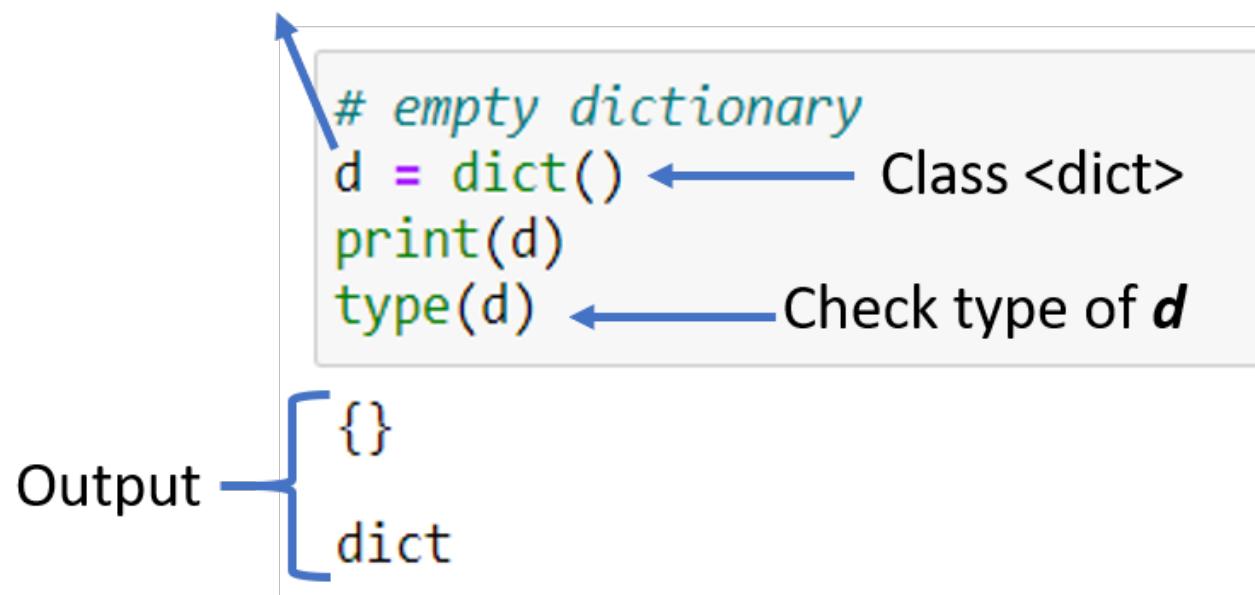
- In dictionaries, the indexes are not by default. (such as 0 in string, list, tuple).
- **{}** is the representation of Dictionary
- Creation of Dictionary
  - An empty dictionary can be created as



# Example-1

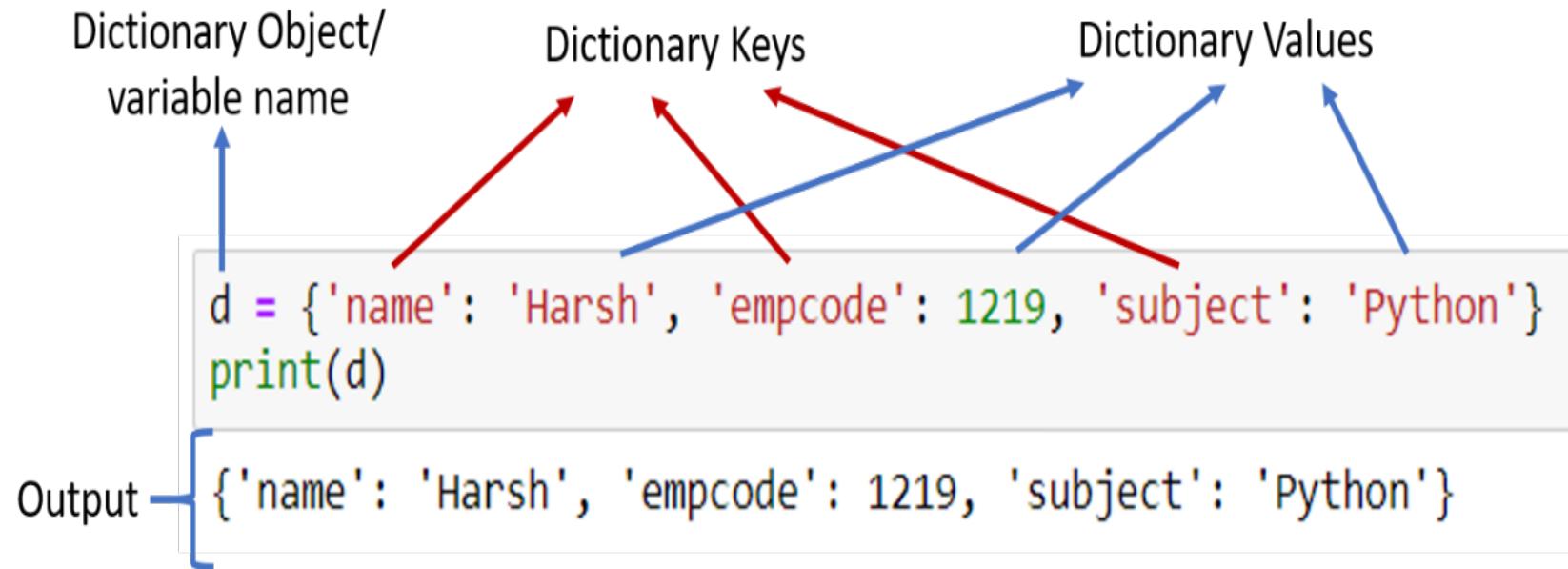
- Creating an empty dictionary

Dictionary Object/  
variable name



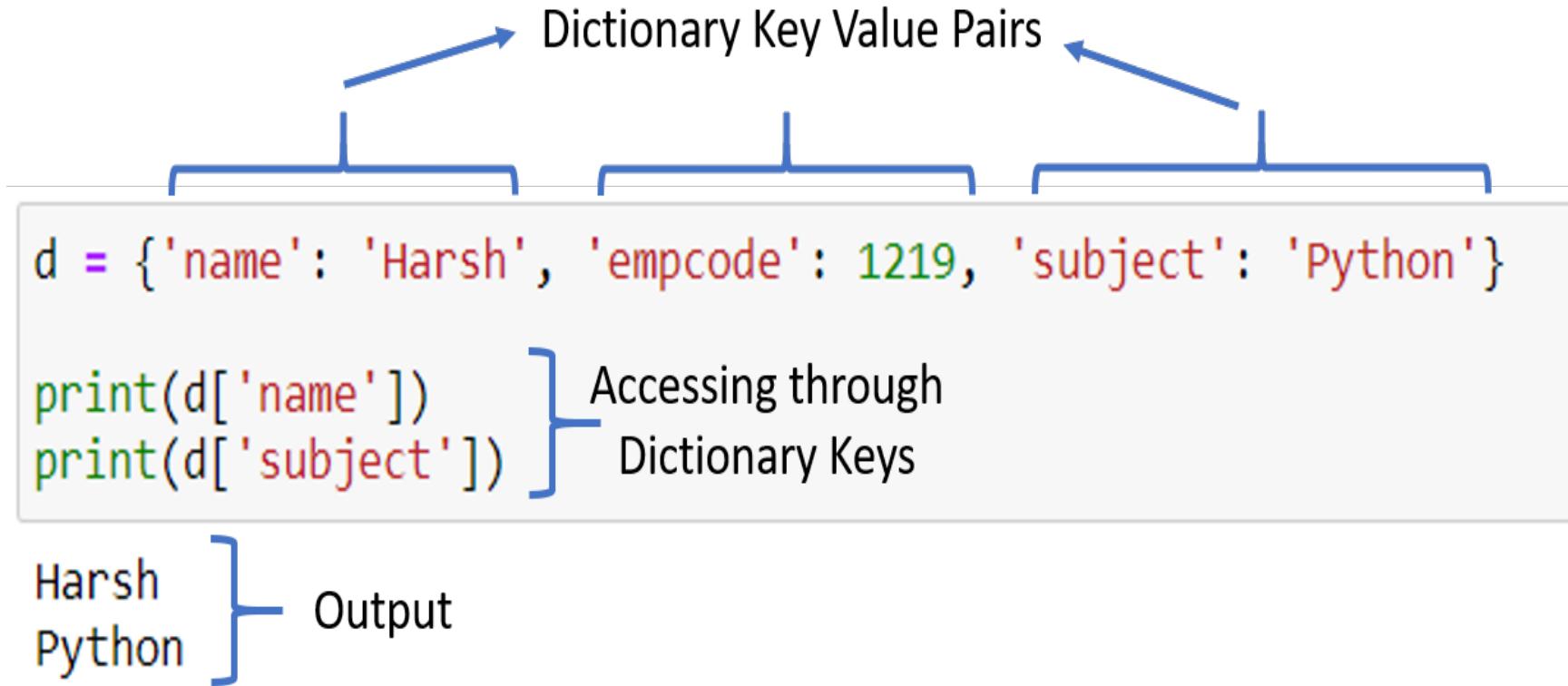
## Example-2

- Creating a non-empty dictionary



# Accessing of Dictionary

- In dictionary, the items are accessed by the keys.



# Can you answer these questions?

1. Output of following code?

```
dict1={'opt1':'Python','opt2':'Java','opt3':'C','opt4':'C++'}  
print(dict1[1])
```

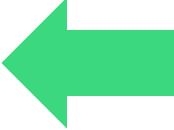
- A) 'Java'
- B) 'Python'
- C) KeyError
- D) None of above



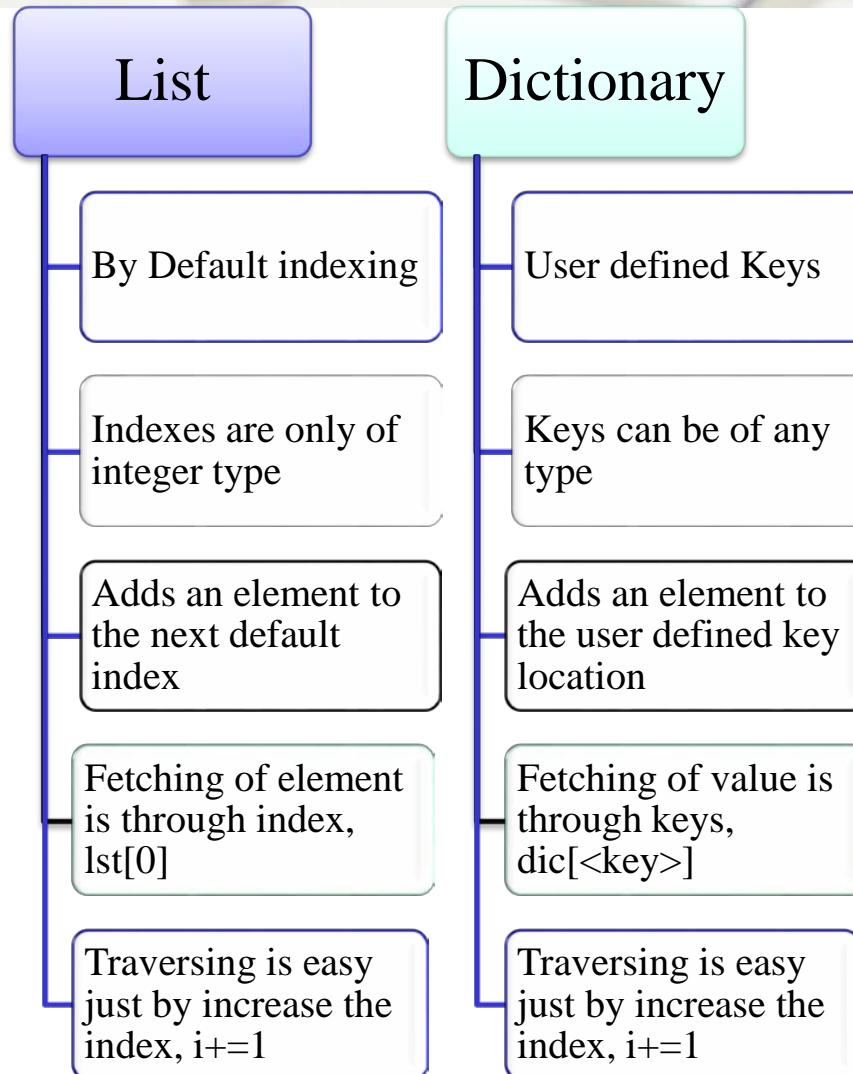
# Can you answer these questions?

2. Output of following code?

```
dict1={1:2,2:4,4:8,8:16,16:32}  
print(len(dict1))
```

- A) 5 
- B) 10
- C) 15
- D) None of above

# Accessing of Dictionary



# Modification in a Dictionary

Dictionary Object/  
variable name

```
d = {'name': 'Harsh', 'empcode': 1219, 'subject': 'Python'}  
print(d['name'])
```

# change the name to full name

```
d['name'] = 'Harsh Khatter' ← Update the Value using Key  
print(d['name'])
```

Harsh

Harsh Khatter

← Value before modification

← Value after modification

} Output

# Can you answer these questions?

1. Output of following code?

A) {1:'Store',2:'Kitchen'}

B) {2:'Store',1:'Kitchen'}

C) KeyError

D) None of above



```
room={1: 'Store', 2: 'Kitchen'}
temp=room[1]
room[1]=room[2]
room[2]=temp
print(room)
```

# Can you answer these questions?

2. Is it possible to change key in dictionary?

A) True

B) False

```
d = {'name': 'Harsh', 'subject': 'Python'}  
print(d['name'])
```

```
d['firstname'] = 'Harsh'  
print(d)
```

When we try to modify the key with same value 'Harsh'

Same values with different keys  
Harsh  
{'name': **'Harsh'**, 'subject': 'Python', 'firstname': **'Harsh'**}

New Key has been created instead of updating the existing key

# Nested Dictionary

- As we have the option of a nested list, similarly, dictionaries can consist of data collections.

```
# Creating a Nested Dictionary
dt = {1: 'Hello', 2: 'to',
      3:{'A' : 'Welcome', 'B' : 'To', 'C' : 2021}}
print(dt)
```

```
{1: 'Hello', 2: 'to', 3: {'A': 'Welcome', 'B': 'To', 'C': 2021}}
```

# Session Plan - Day 7

## 3.4 Dictionary

- Built in Methods in Dictionary
- Loops in Dictionary
- Review Questions
- Practice Exercises

# Built-in Methods in Dictionary

Method	Description
<b>copy()</b>	Copying a dictionary to another dictionary
<b>fromkeys()</b>	Create a new dictionary with key in a data sequence list/tuple with value
<b>get()</b>	If the key is present in the dictionary, its value is returned. If the key is not present in a dictionary, then the default value will be shown as output instead of a KeyError.
<b>items()</b>	this will return the key-value pair as an output.
<b>keys()</b>	this will return only the keys as an output.
<b>values()</b>	this will return only the dictionary values as an output.
<b>update()</b>	this adds the one dictionary with another dictionary.
<b>pop()</b>	The pop() method takes an argument as the dictionary key, and deletes the item from the dictionary.
<b>popitem()</b>	The popitem() method retrieves and removes the last key/value pair inserted into the dictionary.

# copy()

- copy() method provide a fresh copy with different memory location.

```
floor1={1:'Store',2:'Kitchen'}  
floor2=floor1  
floor3=floor1.copy()  
print(floor1)  
print(floor2)  
print(floor3)  
print(id(floor1)==id(floor2))  
print(id(floor1)==id(floor3))
```

## Output

{1: 'Store', 2: 'Kitchen'}
{1: 'Store', 2: 'Kitchen'}
{1: 'Store', 2: 'Kitchen'}
True
False

# fromkey()

```
key=(1,2,3,4)
value='xyz'
sqr={}
sqr.fromkeys(key,value)
```

## Output

```
{1: 'xyz', 2: 'xyz', 3: 'xyz', 4: 'xyz'}
```

# get()

```
std={'name':'Bob','age':24,'phone':[12345,23456]}\nprint(std.get('name'))\nprint(std.get('gender')) # default value None\nprint(std.get('gender','M')) # set default value to 'M'
```

## Output

Bob  
None  
M

# items(),keys(),values()

```
std={'name':'Bob', 'age':24, 'phone':[12345,23456]}
print(std.keys())
print(std.values())
print(std.items())
```

## Output

```
dict_keys(['name', 'age', 'phone'])
dict_values(['Bob', 24, [12345, 23456]])
dict_items([('name', 'Bob'), ('age', 24), ('phone', [12345, 23456])])
```

# update()

```
A={1:1,2:4,4:8,6:36}  
B={3:9,5:25}  
A.update(B)  
print(A)  
print(B)
```

## Output

```
{1: 1, 2: 4, 4: 8, 6: 36, 3: 9, 5: 25}  
{3: 9, 5: 25}
```

# pop(),popitem()

```
A={1:1,2:4,4:8,6:36}  
A.pop(6)  
print(A)
```

{1: 1, 2: 4, 4: 8}

```
A={1:1,2:4,4:8,6:36,8:64}  
A.popitem()  
print(A)
```

**Output**

{1: 1, 2: 4, 4: 8, 6: 36}

# Loops and conditions on dictionaries

- Loops and conditions can easily apply to dictionaries.

```
dict1={1:1,2:4,4:8,6:36}
```

for key in dict1.keys(): print(key)	for val in dict1.values(): print(val)	for item in dict1.items(): print(item)	for i,j in dict1.items(): print(i,":",j)
--	--	---	---

## OUTPUT

1	1	(1,1)	1:1
2	4	(2,4)	2:4
4	8	(4,8)	4:8
6	36	(6,36)	6:36

# Can you answer these questions?

1. Output of following code?

A) {1:1,2:4,3:9,4:16,5:25,6:36}

B) {1:1,2:4,3:9,4:16,5:25} 

C) [1,4,9,16,25]

D) Error

```
newdict={}
for i in range(1,6):
    newdict[i]=i*i
print(newdict)
```

# Can you answer these questions?

1. Output of following code?

A) {1:'Store',2:'Kitchen'}

B) {2:'Store',1:'Kitchen'}

C) KeyError

D) None of above

```
room={1:'Store',2:'Kitchen'}  
temp=room[1]  
room[1]=room[2]  
room[2]=temp  
print(room)
```



# Session Plan - Day 8

## 3.5 Set

- **Creation**
- **Assessing**
- **Modification**
- **Built in methods**
- **Operators**
- **Loops**
- **Frozen Set**
- **Review Questions**

# Set

- A set is another data collection data types in python, which stores unique elements in an unordered way.
- Every element in a set is unique and **immutable(unchangeable)**, i.e. no duplicate values should be there, and ***the values can't be changed.***

# Creation of empty Set

```
# Creation of an Empty set using set()
s1 = set()
print(s1)
print(type(s1))
```

```
set()
<class 'set'>
```

} Output

# Creation of non-empty Set

Elements as Integer

```
s1 = {8,3,15,31,8,10,31,9,2,4}
print(s1)
```

{2, 3, 4, 8, 9, 10, 15, 31} }

Output

```
set1 = {"mango", "banana", "orange"}
count=len(set1)
print(count)
```

3

len() method finds the number of items in set.

String

Integer

Boolean

Float

```
myset = {"name", 2, "age", True , 12.8 }
print(myset)
```

{True, 2, 12.8, 'name', 'age'} }

Output

# Creation of non-empty Set

List of Integers

```
set_int = set([1, 2, 3, 4, 5, 6, 7, 7, 7])  
print(set_int)
```

{1, 2, 3, 4, 5, 6, 7} } Output

String

```
set_str = set("ABESEC")  
print(set_str)
```

{'C', 'B', 'E', 'A', 'S'} } Output

Elements as String

```
set_of_fruits = {'apple', 'orange', 'apple', 'pear', 'orange', 'banana'}  
print(set_of_fruits)
```

{'orange', 'apple', 'pear', 'banana'} } Output

# Can you answer these questions?

2. Is it possible to create an empty set using like s1={} ?

A) True

B) False



```
s1={}
print(type(s1))
<class 'dict'>
```

Output



We can not create  
empty set using {}  
brackets

# Accessing set

- Python set's item cannot be accessed using indexes.

```
set_access = {7, 6, 1, 4, 3, 4, 1, 7}
print(set_access[2])
```

---

```
-----  
TypeError                                Traceback (most recent call last)
<ipython-input-4-68579cc997ed> in <module>
      1 set_access = {7, 6, 1, 4, 3, 4, 1, 7}
----> 2 print(set_access[2])

TypeError: 'set' object is not subscriptable
```

# Built-in Methods in Dictionary

Method	Description
<b>copy()</b>	Copying a set to another set
<b>clear()</b>	Removes all element from set
<b>add()</b>	Adding a new item in set
<b>update()</b>	If the key is present in the dictionary, its value is returned. If the key is not present in a dictionary, then the default value will be shown as output instead of a Key Error.
<b>remove()</b>	to remove the specified element from the given set.
<b>pop()</b>	used to removes a random element from the set and returns the popped (removed) elements.
<b>remove ()</b>	used to remove the specified element from the given set.
<b>discard ()</b>	used to remove the specified item from the given input set. <b><i>the remove() method will give an error if the specified item does not exist but this method will not.</i></b>

# copy()

- copy() method provide a fresh copy with different memory location.

```
set1 = {"mango", "banana", "orange"}  
set2 = set1.copy()  
print(set2)
```

copy() method to copy all set 1 elements to set2

{'orange', 'banana', 'mango'}      } Output

# clear()

```
set1 = {"mango", "banana", "orange"}  
set1.clear()  
print(set1)
```

clear() method to remove all elements

set()  
} Output

# add()

```
# initializing a set
set_add = {21, 23}
print(set_add)

# adding an item
set_add.add(22) ← add() method to add new element
print(set_add)
```

{21, 23}  
{21, 22, 23} } Output

# update()

```
# initializing a set
set1 = {21, 23}
print(set1)

# adding multiple elements
set1.update([22, 13, 14]) ← update() method to add new elements
print(set1)

# adding list and set
set1.update([14, 15], {11, 16, 18}) ← update() method to add new elements
as list and set
print(set1)
```

{21, 23}  
{13, 14, 21, 22, 23}  
{11, 13, 14, 15, 16, 18, 21, 22, 23}

} Output

# remove()

```
set1 = {"mango", "banana", "orange"}  
set1.remove("mango")  
print(set1)
```

{'orange', 'banana'} } Output

remove() method removes any specific element

# pop()

```
set1 = {"mango", "banana", "orange"}  
set1.pop()  
print(set1)
```

pop() method randomly removes any element

{'banana', 'mango'} }

Output

# update()

```
# initializing a set
set1 = {21, 23}
print(set1)
```

```
# adding multiple elements
set1.update([22, 13, 14])
```

update() method to add new elements

```
# adding list and set
set1.update([14, 15], {11, 16, 18})
print(set1)
```

update() method to add new elements  
as list and set

```
{21, 23}
{13, 14, 21, 22, 23}
{11, 13, 14, 15, 16, 18, 21, 22, 23}
```

} Output

# remove()

```
set1 = {"mango", "banana", "orange"}  
set1.remove("mango")  
print(set1)
```

remove() method removes any specific element

{'orange', 'banana'}

} Output

# discard()

```
set1 = {"mango", "banana", "orange"}  
set1.discard("mango") ← discard() method removes any specific element  
print(set1)
```

{'orange', 'banana'} } Output

# Operators

Set Operation	Description	Operator	Method
Union	All unique elements in set1 and set2		union()
Intersection	Elements present in set1 and set2	&	intersection()
Difference	Elements that are present in one set, but not the other	-	difference()
Symmetric Difference	Elements present in one set or the other, but not both	^	symmetric_difference()
Disjoint	True if the two sets have no elements in common	None	isdisjoint()
Subset	True if one set is a subset of the other (that is, all elements of set2 are also in set1)	<=	issubset()
Proper Subset	True if one set is a subset of the other, but set2 and set1 cannot be identical	<	None
Superset	True if one set is a superset of the other (that is, set1 contains all elements of set2)	>=	issuperset()
Proper Superset	True if one set is a superset of the other, but set1 and set2 cannot be identical	>	None

# Union

```
set1 = {'abc', 'pqr', 'xyz'}  
set2 = {'a', 'b'}  
#union  
print(set1 | set2)    ← | operator to join two sets
```

{'a', 'abc', 'b', 'pqr', 'xyz'} } Output

```
set1 = {'abc', 'pqr', 'xyz'}  
set2 = {'a', 'b'}  
#union  
s_union=set1.union(set2) ← Union method to join two sets
```

{'b', 'a', 'xyz', 'abc', 'pqr'} } Output

# Intersection

```
set1 = {'abc', 'pqr', 'xyz'}  
set2 = {'a', 'b', 'abc'}  
#intersection  
print(set1 & set2) ← & operator to intersect two sets
```

{'abc'} } Output

# Intersection

```
set1 = {'abc', 'pqr', 'xyz'}  
set2 = {'a', 'b', 'abc'}  
#intersection  
print(set1 & set2) ← & operator to intersect two sets
```

{'abc'} } Output

```
set1 = {'abc', 'pqr', 'xyz'}  
set2 = {'a', 'b', 'abc'}  
#intersection  
s_intersection=set1.intersection(set2)  
print(s_intersection) ← Intersection method to intersect two sets
```

{'abc'} } Output

# Set Difference

```
set1 = {1, 2, 3, 4, 5}  
set2 = {4, 5, 6, 7, 8}  
print(set1 - set2)
```

- operator to find difference among two sets

{1, 2, 3} } Output

```
set1 = {1, 2, 3, 4, 5}  
set2 = {4, 5, 6, 7, 8}  
diff_set = set1.difference(set2)  
print(diff_set)
```

Difference() method to find difference among two sets

{1, 2, 3} } Output

# Symmetric Difference

```
set1 = {1, 2, 3, 4, 5}  
set2 = {4, 5, 6, 7, 8}  
print(set1 ^ set2)
```

^ operator to find symmetric difference among two sets

{1, 2, 3, 6, 7, 8} } Output

```
set1 = {1, 2, 3, 4, 5}  
set2 = {4, 5, 6, 7, 8}  
symm_diff = set1.symmetric_difference(set2)  
print(symm_diff)
```

Method to find symmetric difference among two sets

{1, 2, 3, 6, 7, 8} } Output

# Loops with set

```
# Creating a set using string
set1 = {"Mango", "Banana", "Orange"}
```

```
# Iterating using for loop
for i in set1: ← Iteration using for loop
    print(i)
```

Banana  
Mango  
Orange

Output

# Frozen Set

A frozen set is a special category of the set which is unchangeable once created

```
# Frozen Set
set1 = {"mango", "banana", "orange"}
set2 = frozenset(set1) ← frozenset() method
```

```
print(type(set1))
print(type(set2))
```

```
<class 'set'>
<class 'frozenset'>
```

} Output

# Frozen Set

```
set1 = {"Mango", "Banana", "Orange"}  
set2 = frozenset(set1)  
#add new element to frozen set  
new="Grapes"  
print(set2.add(new))
```

Trying to add new element in frozen set using `add()` method

AttributeError Traceback (most recent call last)  
<ipython-input-36-a4b367bff942> in <module>  
 3 #add new element to frozen set  
 4 new="Grapes"  
----> 5 print(set2.add(new))

AttributeError: 'frozenset' object has no attribute 'add'

Output

# Summary

List	Tuple	Set	Dictionary
<i>List is a collection of values that is ordered.</i>	<i>Tuple is ordered and unchangeable collection of values</i>	<i>Set stores the unique values as data collection</i>	<i>Dictionary is a collection of key-value pairs</i>
<i>Represented by [ ]</i>	<i>Represented by ()</i>	<i>Represented by { }</i>	<i>Represented by { }</i>
<i>Duplicate elements allowed</i>	<i>Duplicate elements allowed</i>	<i>Duplicate elements not allowed</i>	<i>Duplicate keys not allowed, in dictionary values allowed duplicate</i>
<i>Values can be of any type</i>	<i>Values can be of any type</i>	<i>Values can be of any type</i>	<i>Keys are immutable type, and value can be of any type</i>
<i>Example: [1, 2, 3, 4]</i>	<i>Example: (1, 2, 3, 4)</i>	<i>Example: {1, 2, 3, 4}</i>	<i>Example: {1:1, 2:2, 3:3, 4:4}</i>
<i>List is mutable</i>	<i>Tuple is immutable</i>	<i>Set is mutable</i>	<i>Dictionary is mutable</i>
<i>List is ordered</i>	<i>Tuple is ordered</i>	<i>Set is unordered</i>	<i>Dictionary is insertion ordered</i>
<i>Creating an empty list <code>l=list()</code> <code>l = []</code></i>	<i>Creating an empty Tuple <code>t = tuple ()</code> <code>t=()</code></i>	<i>Creating a set <code>s=set ()</code></i>	<i>Creating an empty dictionary <code>d=dict()</code> <code>d={}</code></i>

# Can you answer these questions?

1. Which of the following Python code will create a set?

- (i) `set1=set((0,9,0))`
- (ii) `set1=set([0,2,9])`
- (iii) `set1={}`

- A) iii
- B) i and ii
- C) ii and iii
- D) All of Above



# Can you answer these questions?

2. What is the output of following code?

```
set1=set((0,9,0))  
print(set1)
```

- A) {0,0,9}
- B) {0,9}
- C) {9}
- D) Error



# Can you answer these questions?

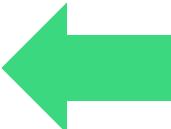
3. What is the output of following python code?

```
set1={1,2,3}
```

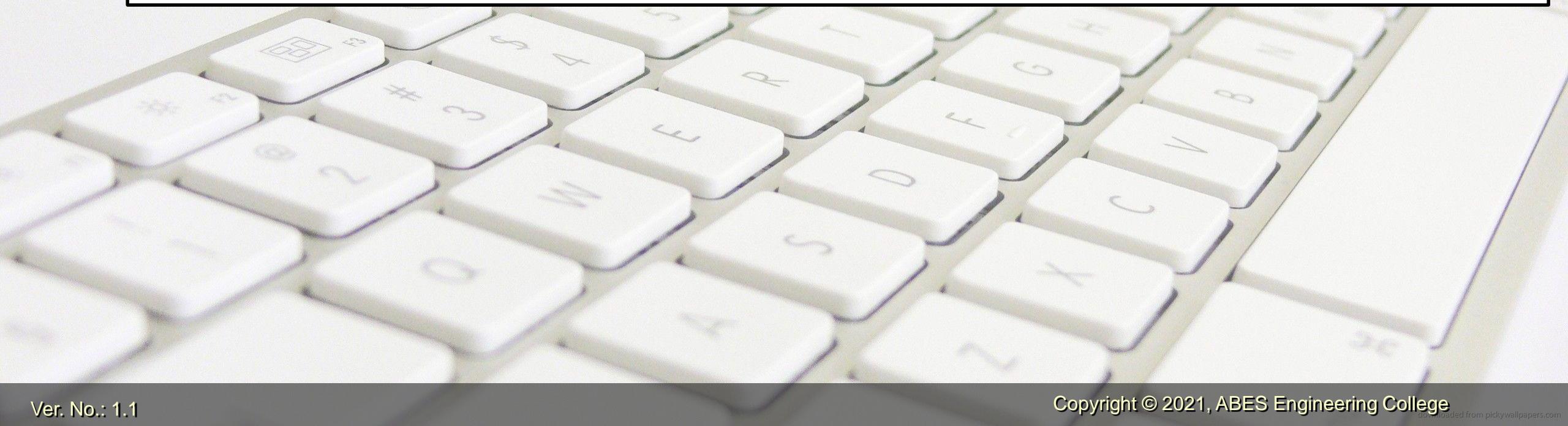
```
set1.add(4)
```

```
set1.add(4)
```

```
print(set1)
```

- A) {1,2,3}
- B) {1,2,3,4} 
- C) {1,2,3,4,4}
- D) Error

## 4. Functions and Module



# General Guideline

© (2021) ABES Engineering College.

This document contains valuable confidential and proprietary information of ABESEC. Such confidential and proprietary information includes, amongst others, proprietary intellectual property which can be legally protected and commercialized. Such information is furnished herein for training purposes only. Except with the express prior written permission of ABESEC, this document and the information contained herein may not be published, disclosed, or used for any other purpose.

# Objective of Functions and Modules

**To describe the importance of functions in Python**

**To explain the function definition in Python**

**To develop python programs with functions**

**To use the void functions and return statements**

**To explain the difference between different function argument types and use them**

**To select built-in functions in Python to write programs in Python.**

# Topics Covered

## Day 1

### 4.1 Introduction to Functions

- 4.1.1 Arguments and parameters
- 4.1.2 Return statement

## Day 2

### 4.1 Introduction to Functions

- 4.1.3 Types of Function arguments
- 4.1.4 Scope and Lifetime of variables

## Day 3

### 4.2 Anonymous and Higher Order Function

- 4.2.1 Anonymous Function

## Day 4

### 4.2 Anonymous and Higher Order Function

- 4.2.2 First Class Function and Higher Order Function

# Topics Covered

Day 5

## 4.3 Modules

- 4.3.1 Creation of module
- 4.3.2 Importing module
- 4.3.3 Standard Built-In Module

Day 6

## 4.4 Iterative Built-in Functions

# Session Plan - Day 1

## 4.1 Introduction to functions

**4.1.1 Argument and parameters**

**4.1.2 Return statement**

**4.1.3 Types of function arguments**

**4.1.4 Scope and lifetime of variables.**

# Introduction

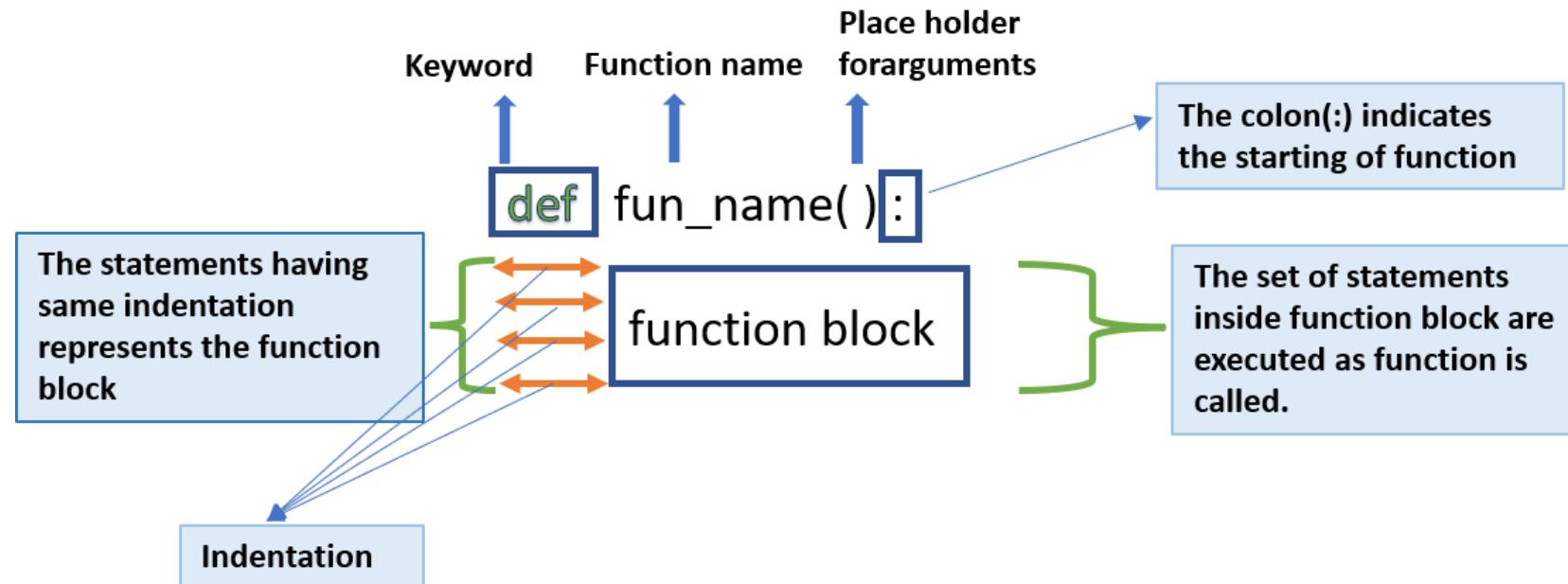
A large program is divided into small blocks of code called **functions**, to:

- Increase the reusability of code.
- Ease of programming.

**Function is a group of all statement(activities) that perform a specific task.**

# Syntax

The syntax of **function definition** is:



# Benefits of Using Functions:

- Avoid **duplication** of the similar type of codes
- Increases Program **readability** and Improved Concept Clarity
- Divide the bigger problem into **smaller chunks**
- **Reusability** of code

# Example

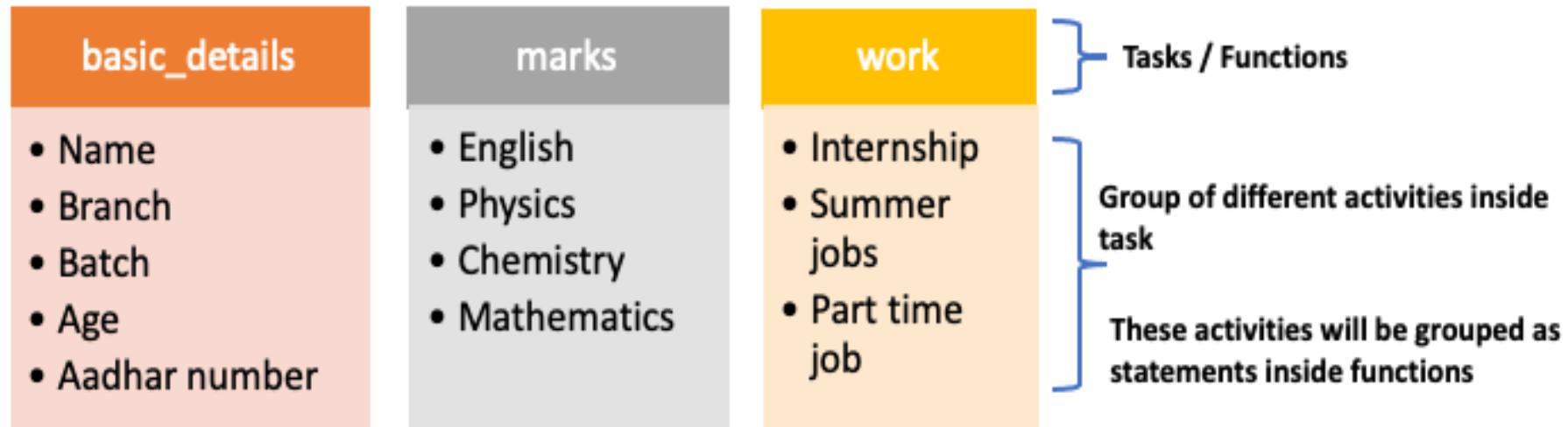
In **real world scenario**, we need to group all the activities related to a specific task, like:

1. A task of maintaining **student's basic details** involves a group of different activities like:  
***Collecting their name, branch, batch, age, aadhar number, etc.***
2. Similarly, the task of **maintaining marks of students** involves a group of different activities like: ***Collecting marks of English, Physics, Chemistry, Mathematics, etc.***

# Contd..

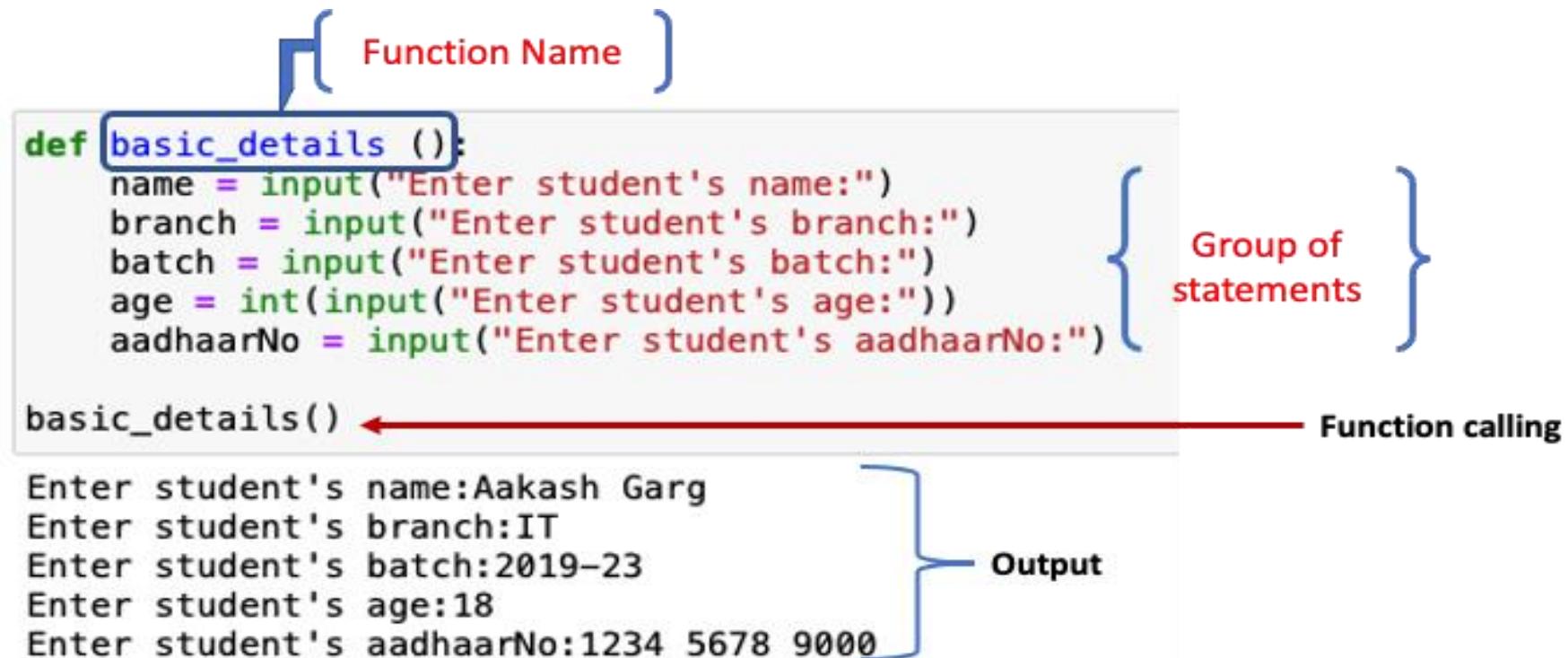
➤ We can create three functions like:

- basic\_details ()
- marks()
- work()



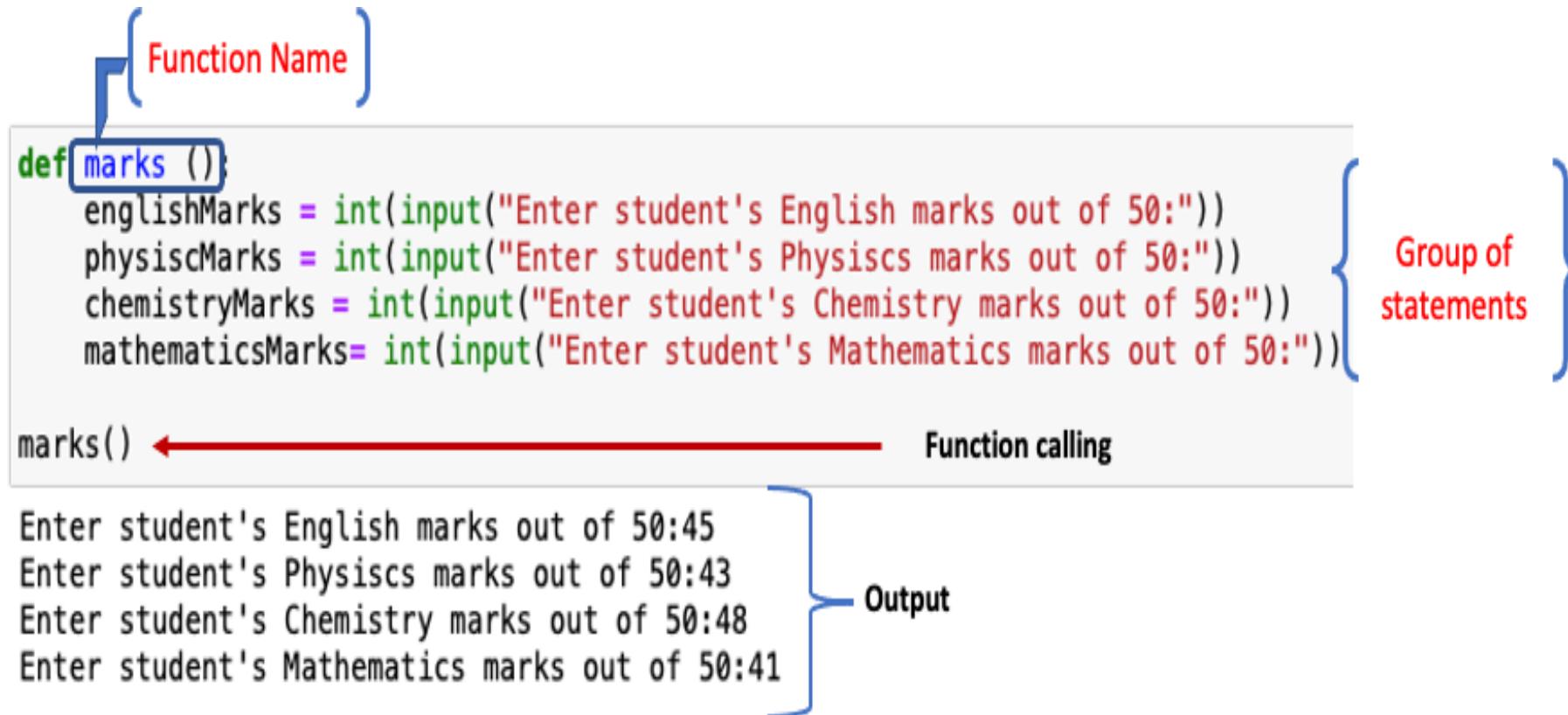
# Contd..

## Function 1: basic\_details ()



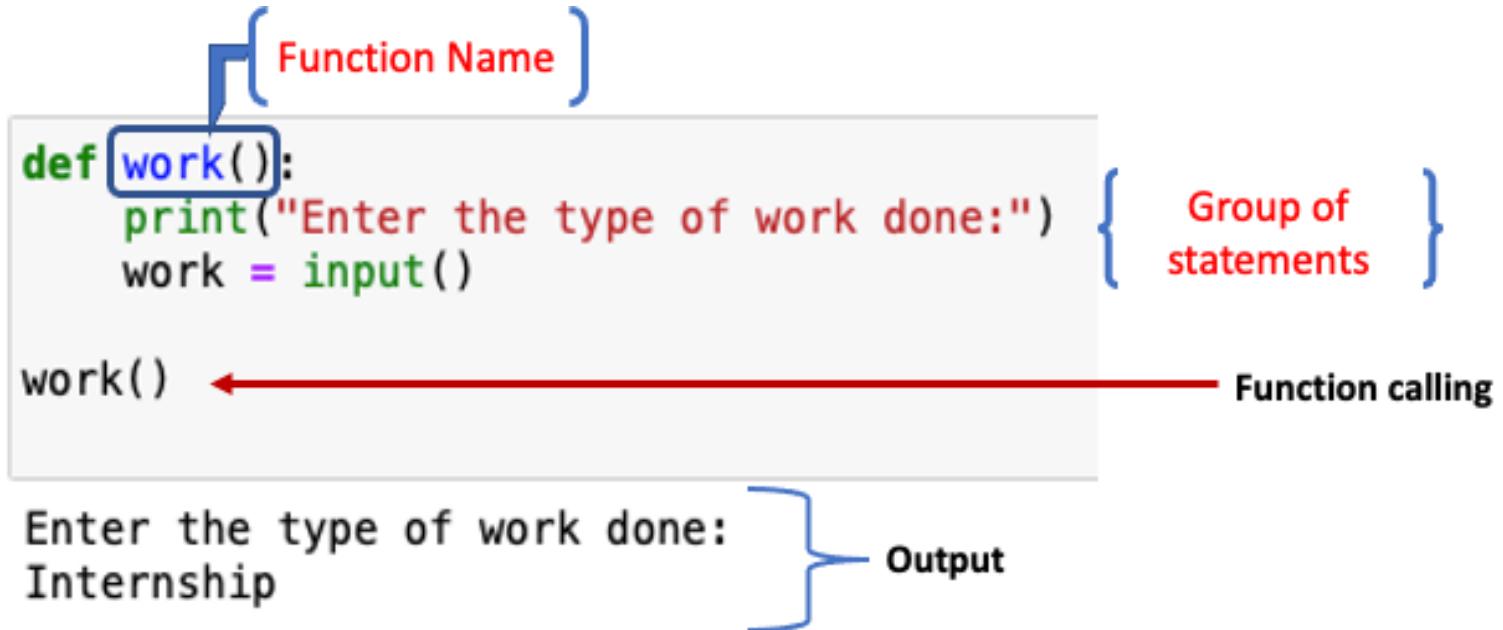
# Contd..

## Function 2: marks ()



# Contd..

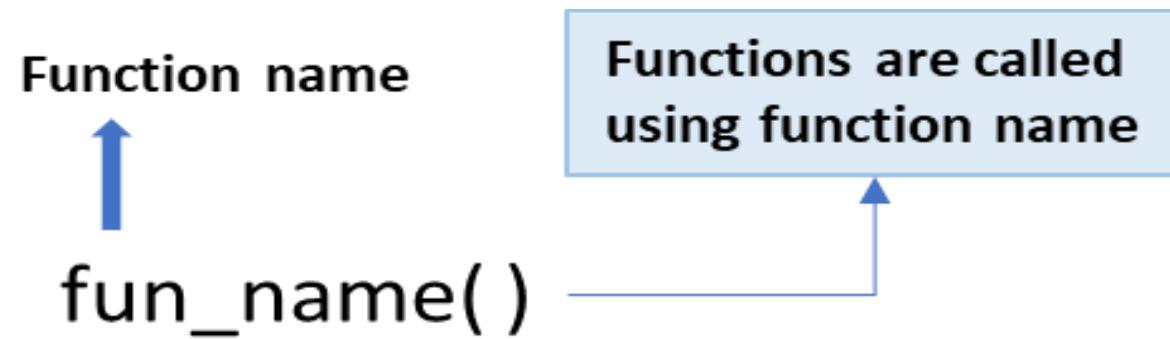
## Function 3: work ()



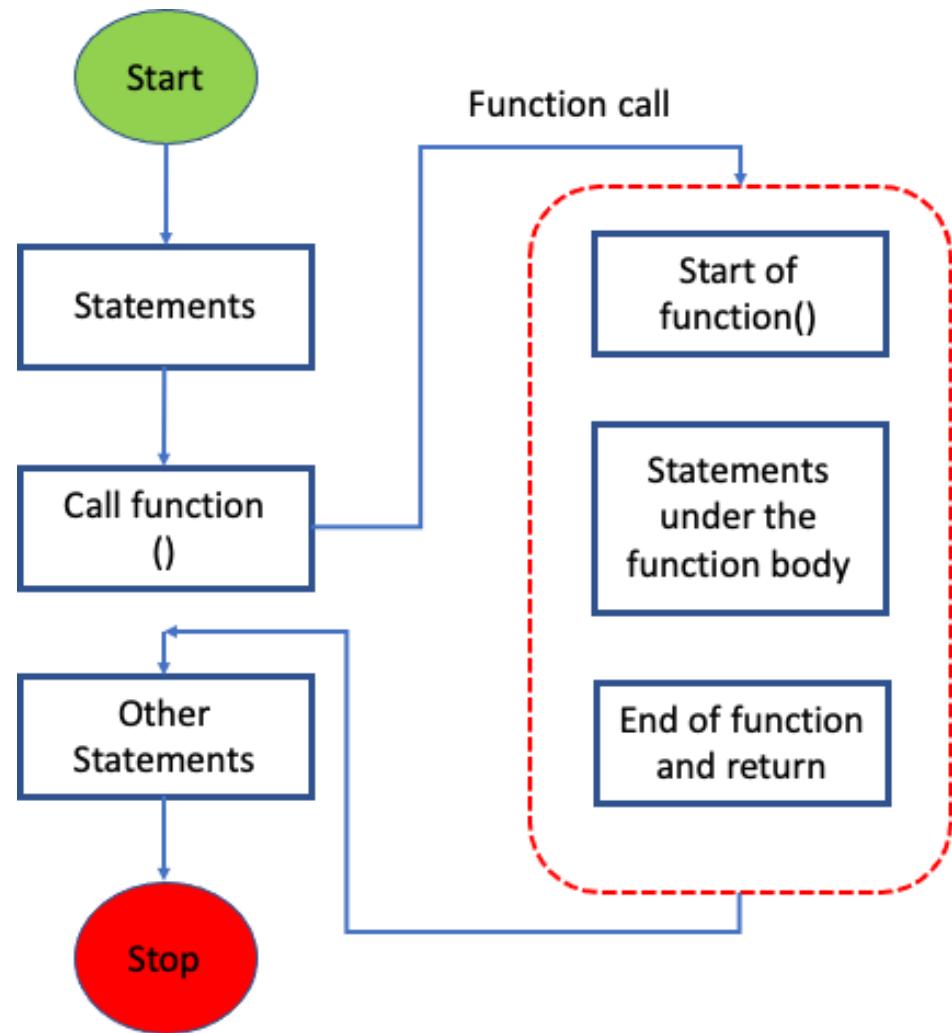
# Function Call

A function is called using function name through **calling environment**.

The syntax of function call is:



# Flow chart



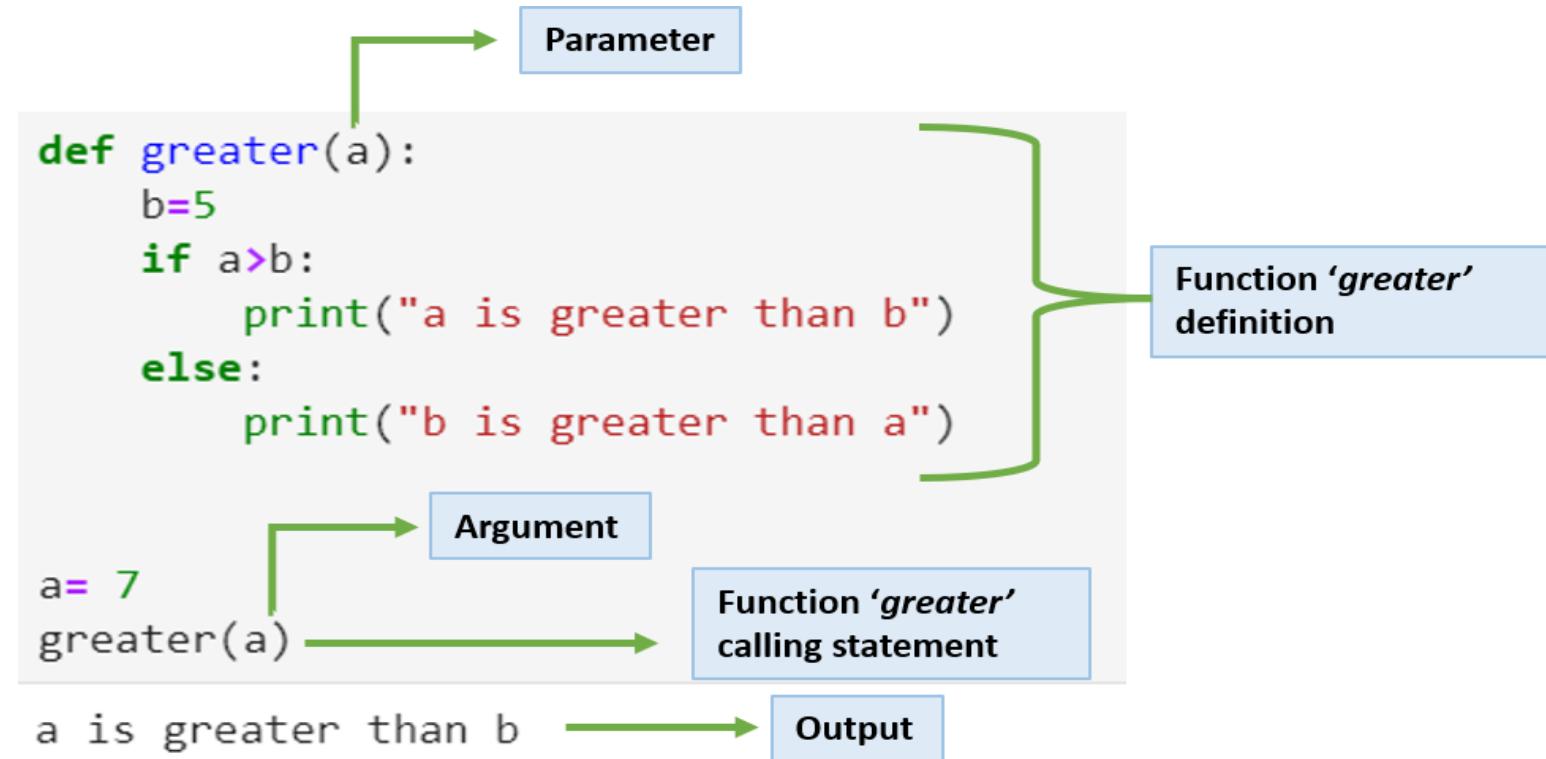
# Arguments and parameters

- Arguments are used to **pass the information** to the functions.
- They are mentioned after function name inside the parentheses in **function calling statement**.
- **Any number of arguments** can be added by separating them by commas.

**Note:** The number of parameters in function definition must be same as the number of arguments in function calling statement.

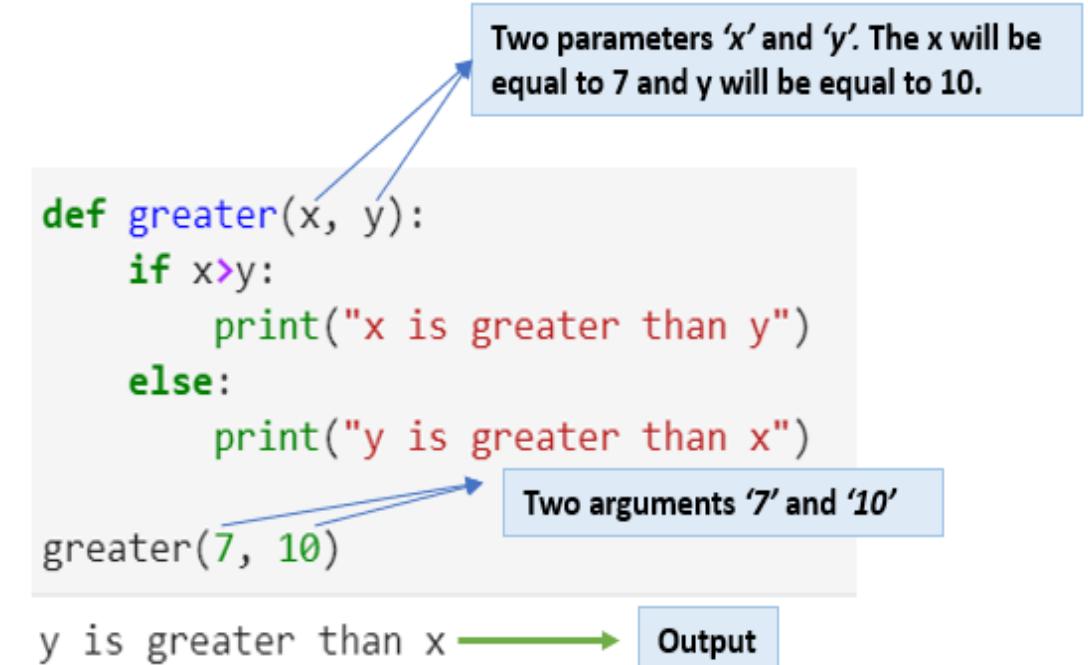
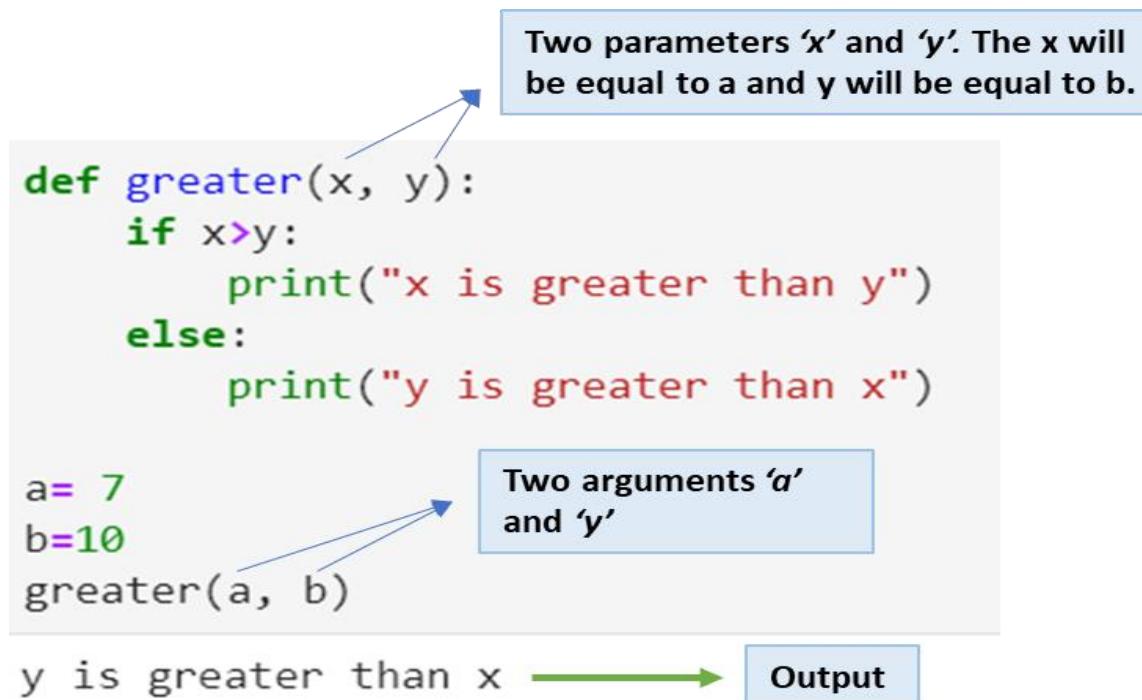
# Example 1

Write a python program to determine the greater number among two numbers using function *greater(a)*, by passing one number as argument in function calling.



# Example 2

Write a python program to determine the greater number among two numbers by passing both the number as argument in function calling.

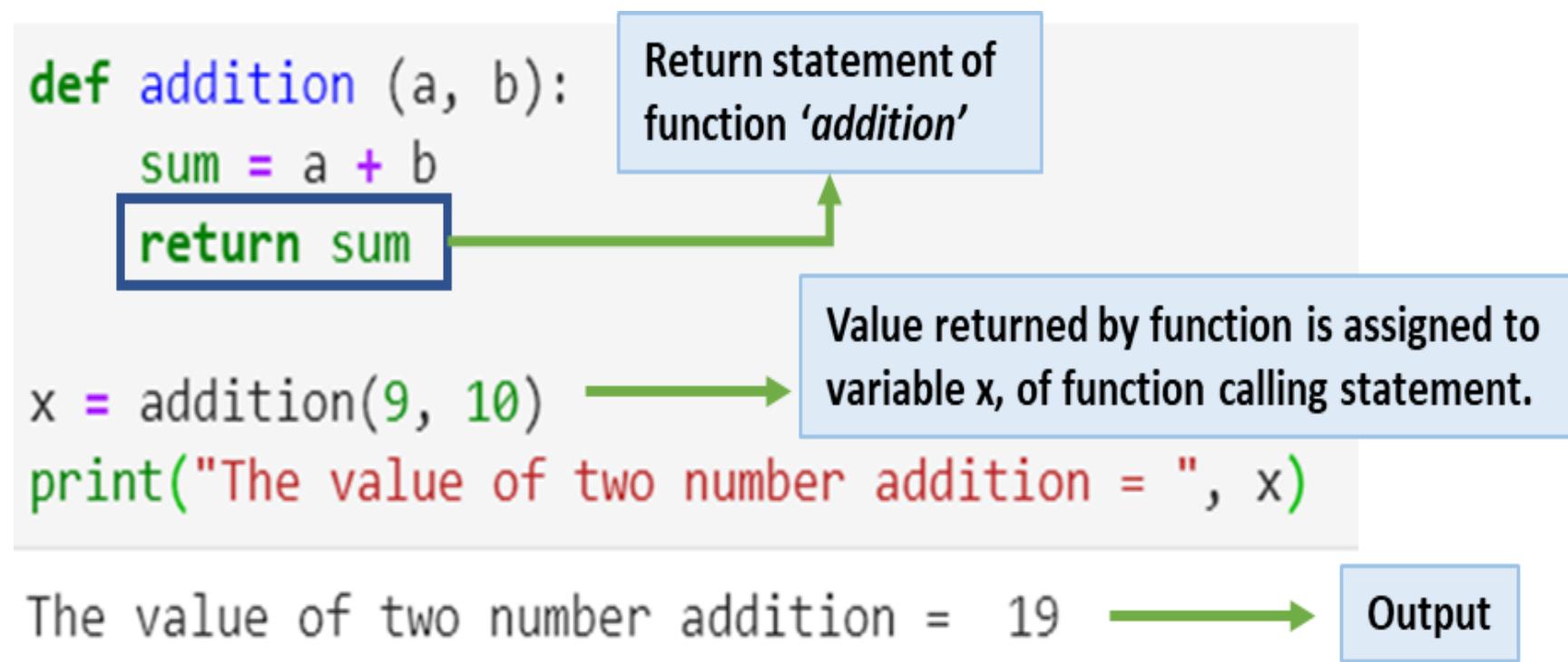


# Return statement

- Return statement indicate the **end of function execution**.
- It is also used to **return the values or results** to the function calling statement.

# Example 1

Write a python program to demonstrate the return statement of function for the addition of two numbers.



# Review Questions

1. Which keyword is used for function?
  - a) Fun
  - b) Define
  - c) Def
  - d) Function



2. What will be the output of the following Python code?

```
def f(x, y, z): return x + y + z
f(2, 30, 400)
```

- a) 432
- b) 24000
- c) 430
- d) No output



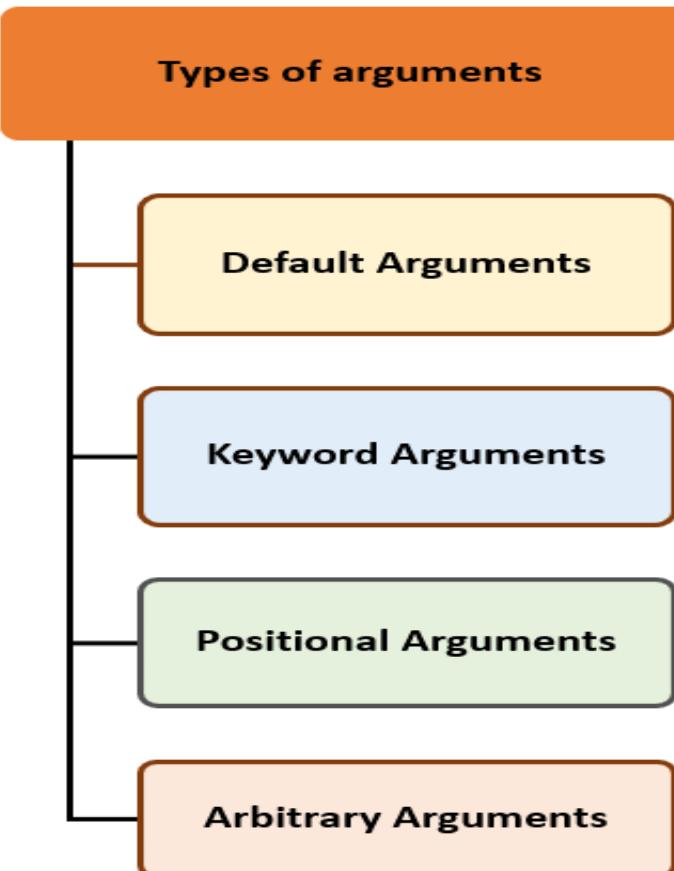
# Session Plan - Day 2

## 4.1 Introduction to functions

- **4.1.3 Types of Function arguments**
- **4.1.4 Scope and Lifetime of variables**

# Types of Function arguments

There are five types of arguments in Python function:



# Default arguments :

Default arguments are values that are provided in the function definition.

## Syntax:

```
def function_name(parameter1, parameter2=value2, parameter3=value3):
```

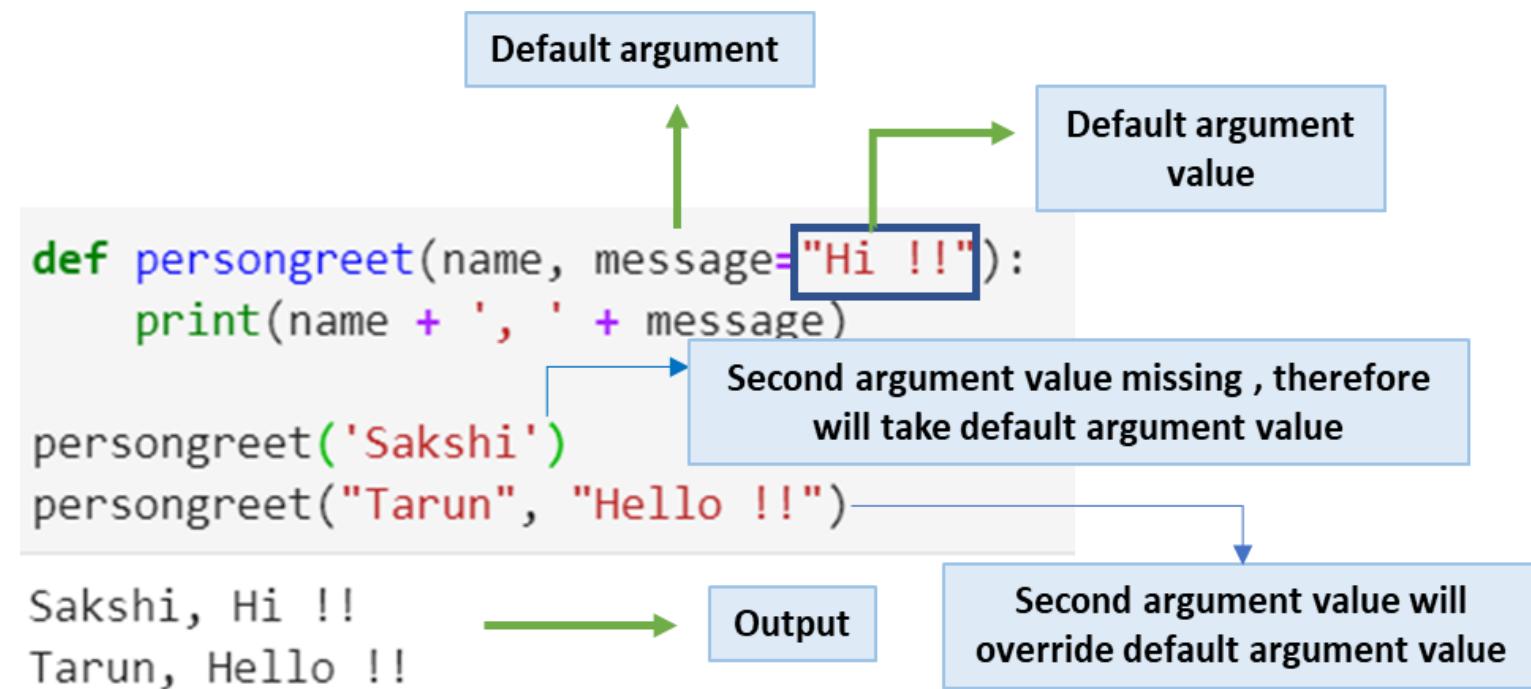


The diagram consists of two blue arrows pointing upwards from the text "Default value" to the assignment operators (=) in the Python code. The first arrow points to the second parameter, and the second arrow points to the third parameter.

**Note:** Function can have any number of default arguments. If the values are provided to the default arguments during the function calling, it overrides the default value.

# Example 1

Write a python program to greet a person with person name and message provided in function calling statement. If message not provided in function call, then print a default message “Hi !!”.



# Keyword arguments :

Default arguments are keyword arguments whose values are assigned at the time of function definition.

**Syntax:**

```
def function_name(parameter1, parameter2)
    return(parameter1+parameter2)
```

```
function_name(parameter1=value1,parameter2=value2) → Calling Environment
```

Keyword value

Keyword value

# Example 1

Write a program to demonstrate keyword arguments.

```
def add(a,b,c):  
    return(a+b+c)  
  
add(b=10,c=15,a=20)
```

b, c, a are keyword arguments

```
def add(a=20,b,c):  
    return(a+b+c)  
  
add(b=10,c=15,a)
```

An error because it uses the default argument after a keyword argument

```
File "<ipython-input-10-8a31cfab86a3>", line 1  
    def add(a=20,b,c):  
               ^  
SyntaxError: non-default argument follows default argument
```

45 } Output

} Output

# Positional arguments :

Positional arguments are arguments that need to be included in the proper position or order while calling the function.

**Syntax:**

```
def function_name(parameter1, parameter2)
    return(parameter1+parameter2)

function_name(value1,value2)
```

# Example 1

Write a program to demonstrate positional arguments.

```
def add(a,b,c):  
    return (a+b+c)  
print (add(10,20,30))
```

10,20,30 are the values for  
Positional Arguments a,b,c

60 } Output

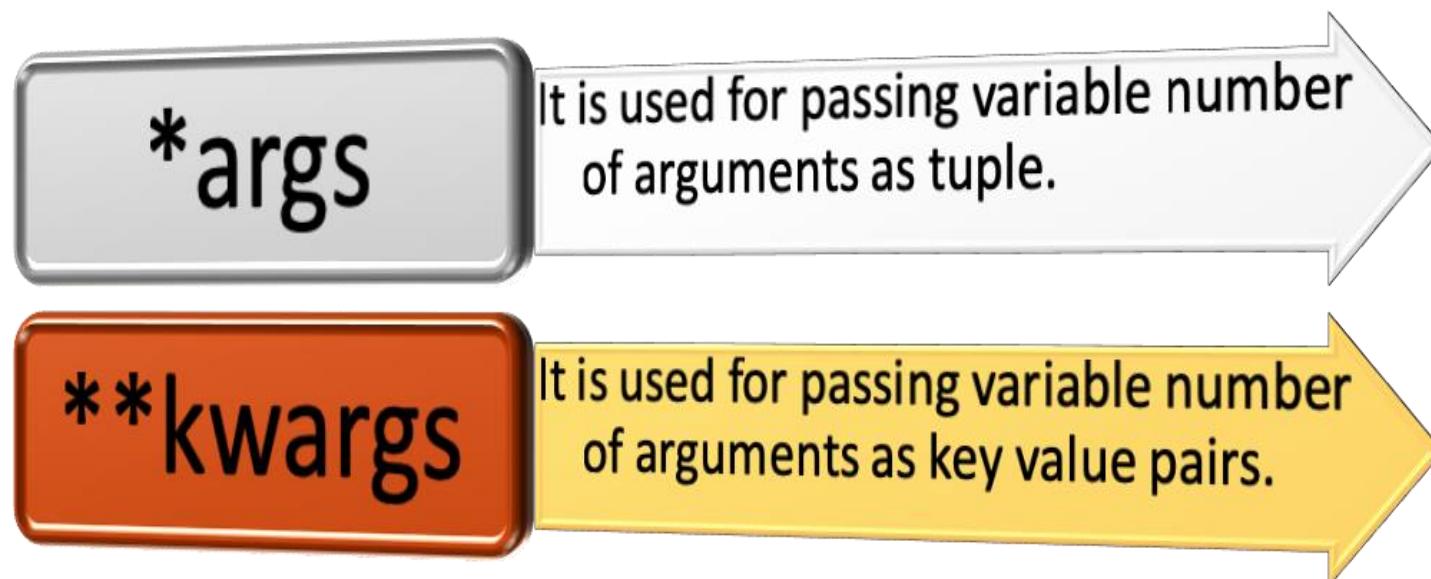
# Arbitrary Arguments

- When we are **not sure** in the advance that **how many arguments our function would require.**
- This kind of situation is handled through function calls with an arbitrary number of arguments.

**Note:** We define the arbitrary arguments while defining a function using the asterisk (\*).

# Types of Arbitrary Arguments

There are **two** types of arbitrary arguments as illustrated in the given figure below.



# \*args

The special syntax \*args in function definitions is used to pass the **variable number of arguments to a function.**

**Note:** Generally, \* args is used by convention but you can take any variable for example \*abc will work in same way as \* args.

# Example 1

Write a program to demonstrate \*args arbitrary arguments.

```
def fruits(*args): # args is a tuple with arguments
    for fruit in args:
        print(fruit)

fruits("Orange", "Banana", "Apple", "Grapes")
```

Output

Orange  
Banana  
Apple  
Grapes

A blue curved arrow points from the text "# args is a tuple with arguments" down to the line "fruits("Orange", "Banana", "Apple", "Grapes")". A red oval highlights the argument list "fruits("Orange", "Banana", "Apple", "Grapes")". A blue bracket labeled "Output" groups the four printed fruit names.

# \*\*kwargs

The `**kwargs` function in python is used to **pass an arbitrary number of keyword arguments** called `kwargs`.

**Note:** `**` (Double star allows us to pass variable number of keyword arguments). This turns the identifier-keyword pairs into a dictionary within the function body.

# \*\*kwargs

Write a program to demonstrate \*\*kwargs arbitrary arguments.

```
def fruits(**kwargs): # kwargs is a dictionary with key value pairs
```

```
    print(kwargs)
    for fruit in kwargs.values()
        print(fruit)
```

```
fruits(fruit1= "Orange",fruit2= "Banana",fruit3= "Apple",fruit4= "Grapes")
```

```
{'fruit1': 'Orange', 'fruit2': 'Banana', 'fruit3': 'Apple', 'fruit4': 'Grapes'}
```

```
Orange
```

```
Banana
```

```
Apple
```

```
Grapes
```

Output

# \*\*kwargs

Write a program using this function to generate perfect numbers from 1 to 1000. [An integer number is said to be “perfect number” if its factors, including 1(but not the number itself), sum to the number. E.g., 6 is a perfect number because  $6=1+2+3$ ].

```
def perfect_num(n):
    sum = 0
    for i in range(1,n):
        if n%i == 0:
            sum = sum + i
    if(sum == n):
        return True
    else:
        return False
for i in range(1,1001):
    if(perfect_num(i)):
        print(i)
```

Function returning true or false based on perfect number condition

6  
28  
496

Output

# Scope and Lifetime of variables

- Let's consider a situation that you book a train ticket in sleeper coach and you got berth in S1 coach and 48 seat number.
- Now your ticket is valid for mentioned train and coach only this is called **scope** of your ticket.
- Your ticket would expire after mentioned journey date and that is called **lifetime**.

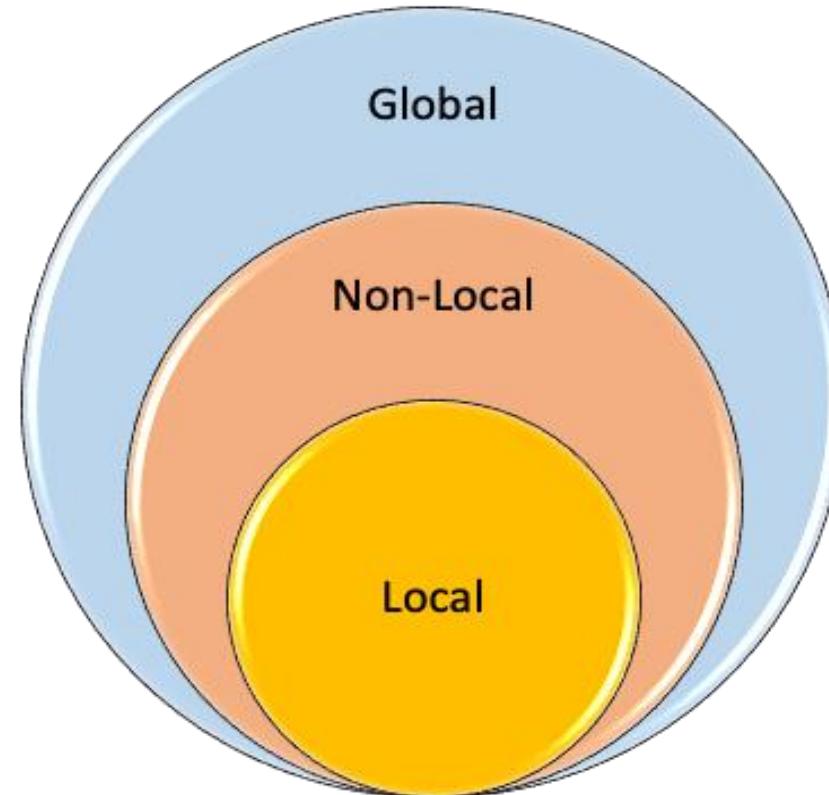
# Scope and Lifetime of variables

- **Scope:** The scope of a variable determines its **accessibility and availability** in different portions of a program. Their availability depends on where they are defined.
- **Lifetime:** The lifetime of a variable is the time during which the variable **stays in memory**.

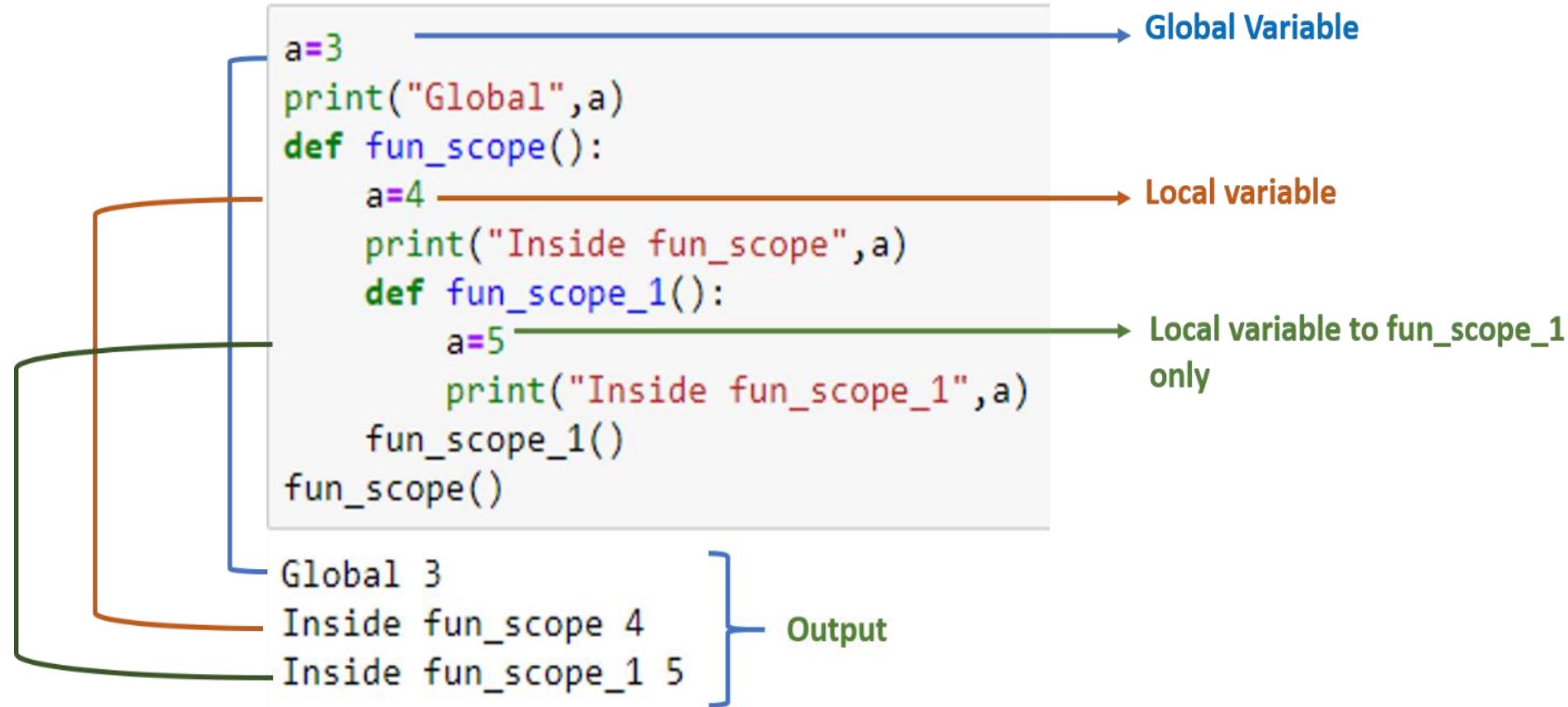
# Types of variables

Three kinds of variables in Python:

- Global Variable
- Local Variables
- Non local variables



# Demonstration of Local and Global



# Global Variables

Variables that are **declared outside of a function** are known as global variables.

**Example:** Write a program to create global variable in python.

```
global_variable = "ABES Engineering College" } Global variable declaration
def welcome():
    print("Welcome to :",global_variable)

welcome() ← Function Calling
Welcome to : ABES Engineering College } Output
```

# Local Variables

- These are variables which are **defined inside a function**.
- Their scope is only to that function.
- They cannot be accessed from the outside of the function.

**Example:** Write a program to create local variable in python.

```
def welcome():
    local_variable = "ABES Engineering College"
    print("Welcome to :", local_variable)
```

welcome() ← Function Calling

Welcome to : ABES Engineering College } Output

# Example: Local and global variable in one program.

```
college_name = "ABES Engineering College" } Global variable declaration
def welcome():
    college_name = "ABES Engineering College, Ghaziabad" } local variable declaration
    print("Welcome to :", college_name)
```

welcome() ← Function will print local variable

print(college\_name) ← Function will print global variable

Welcome to : ABES Engineering College, Ghaziabad } Output
ABES Engineering College

# Example: To modify the variable outside of the current scope.

```
college_name = "ABES Engineering College" } Global variable declaration
def welcome():
    college_name = "ABES Engineering College, Ghaziabad" } local variable declaration
    print("Welcome to :", college_name)

welcome() ← Function will print local
print(college_name) ← variable Function will print global
                                variable

Welcome to : ABES Engineering College, Ghaziabad } Output
ABES Engineering College
```

# Non - Local Variables

- Nonlocal variables are used in nested functions.
- When a variable is either in local or global scope, it is called a nonlocal variable.

**Note:** Nonlocal keyword will prevent the variable from trying to bind locally first, and force it to go a level 'higher up'.

# Non - Local Variables

- Nonlocal variables are used in nested functions.
- When a variable is either in local or global scope, it is called a nonlocal variable.

**Note:** Nonlocal keyword will prevent the variable from trying to bind locally first, and force it to go a level 'higher up'.

# Example 1

A program to create local variables for nested functions.

```
def welcome():
    college_name = "ABES Engineering College"
    def welcome_again():
        college_name = "ABES Engineering College, Ghaziabad"
        university_name="AKTU"
        print("Welcome to :",university_name)

    welcome_again()
    print("Welcome to :",college_name)

welcome() ←
```

Local variable for `welcome()`

local variable for `welcome_again()`

Function will print local variable `college_name` from `welcome()`

Output

Welcome to : AKTU  
Welcome to : ABES Engineering College

# Example 2

A program to use nonlocal keyword for nested functions.

```
def welcome():
    college_name = "ABES Engineering College" } Local variable for welcome()

    def welcome_again(): {Keyword}
        nonlocal college_name
        college_name = "ABES Engineering College,Ghaziabad" } local variable for
        university_name="AKTU"
        print("Welcome to :",university_name)

    welcome_again()
    print("Welcome to :",college_name)

welcome() ←
Welcome to : AKTU
Welcome to : ABES Engineering College,Ghaziabad } Output
```

Function will print local variable **college\_name** from **welcome\_again()**

# Review Questions

What will be the output of the following Python code?

```
def cube(x):  
    return x * x * x  
x = cube(3)  
print(x)
```

- a) 9
- b) 3
- c) 27
- d) 30



What will be the output of the following Python code?

- a) 9
- b) 27
- c) None of the above
- d) Both of the above

```
def power(x, y=2):  
    r = 1  
    for i in range(y):  
        r = r * x  
    return r  
print(power(3))  
print(power(3, 3))
```



# Session Plan - Day 3

## 4.2 Anonymous and Higher Order Function

### 4.2.1 Anonymous Function

#### 4.2.1 Lambda Function

# Anonymous Function

- Sometimes we only have **single expression to evaluate**, so rather than creating regular function we create anonymous Function.
- These functions **do not have return statement**, because by default evaluated single expression is returned.
- This is **one liner code** and have following properties –
  - Function without name
  - One expression statement but any number of arguments.

# Syntax

Keyword



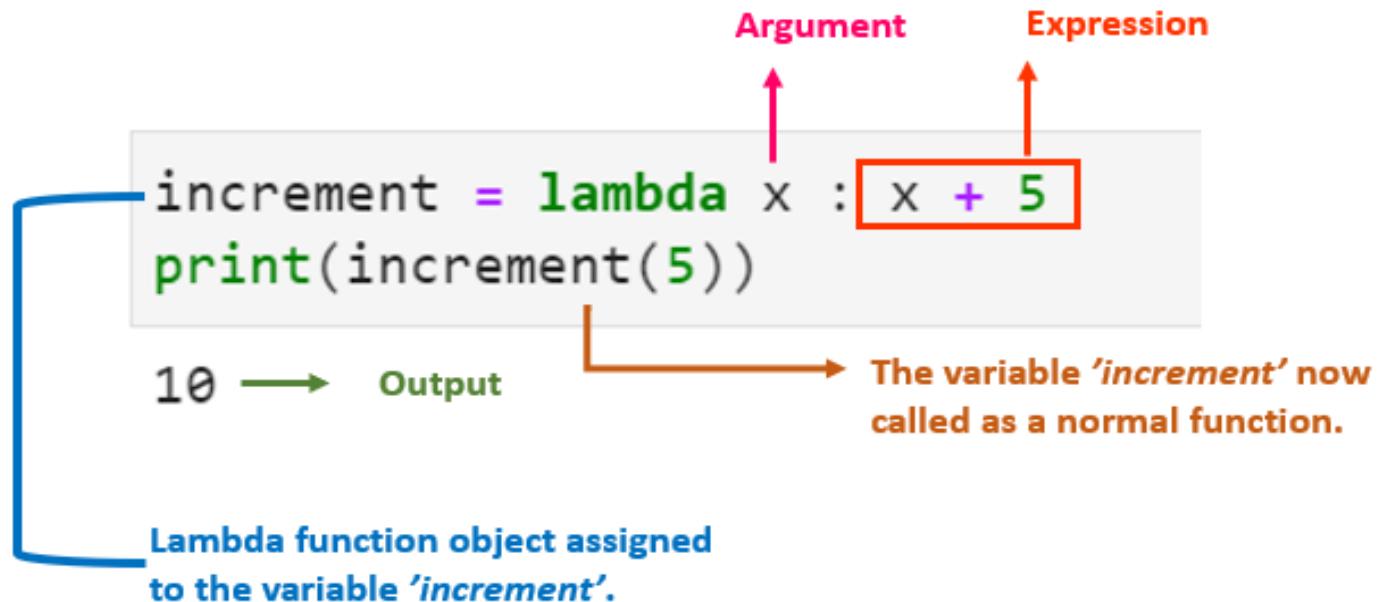
Colon



lambda arguments : expression

# Scenario 1

Suppose we want to increment the value of variable by five, without writing lengthy code we can use lambda function.



# Example 1

A program to add three values, using multiple arguments in lambda function.

## Multiple arguments

```
sum = lambda a, b, c : a + b + c
print(sum(5, 6, 7))
```

18 → Output

## Example 2

A program using lambda function to calculate the simple interest.

```
SimpleInterest = lambda P, R, T: P * R * T
print(SimpleInterest(400, 6, 2 ))
```

4800 → Output

# Review Questions

1. Python supports the creation of anonymous functions at runtime, using a construct called \_\_\_\_\_

- a) lambda
- b) pi
- c) anonymous
- d) none of the mentioned



## 2. What will be the output of the following Python code?

```
y = 6
z = lambda x: x * y
print z(8)
```

- a) 48
- b) 14
- c) 64
- d) None of the mentioned



# Session Plan - Day 4

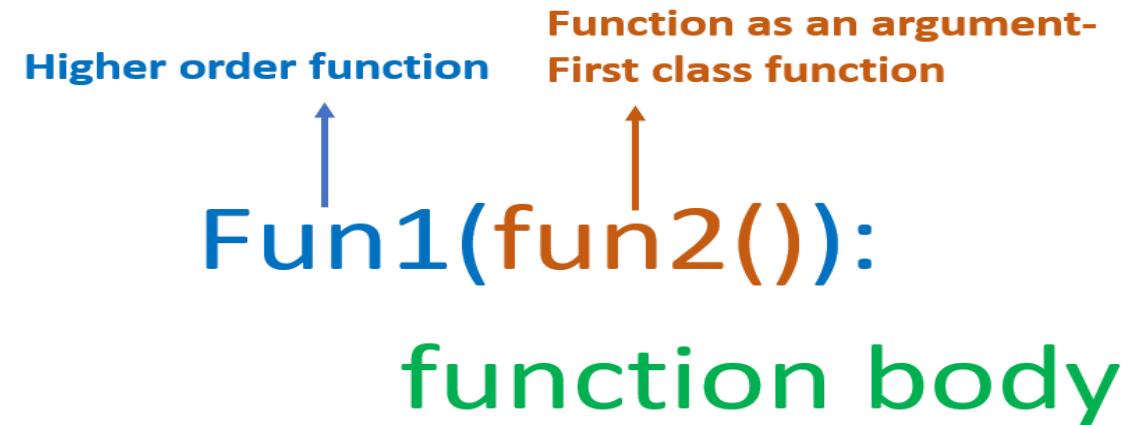
## 4.2 Anonymous and Higher Order Function

- **4.2.2 First Class Function and**
- **4.2.2 Higher Order Function**

# First Class Function and Higher Order Function

- In first class function, **function which can be passed as an argument**.
- Functions which take first class **function as an argument** are called Higher order function.

Syntax:



# Properties of First-Class Function:

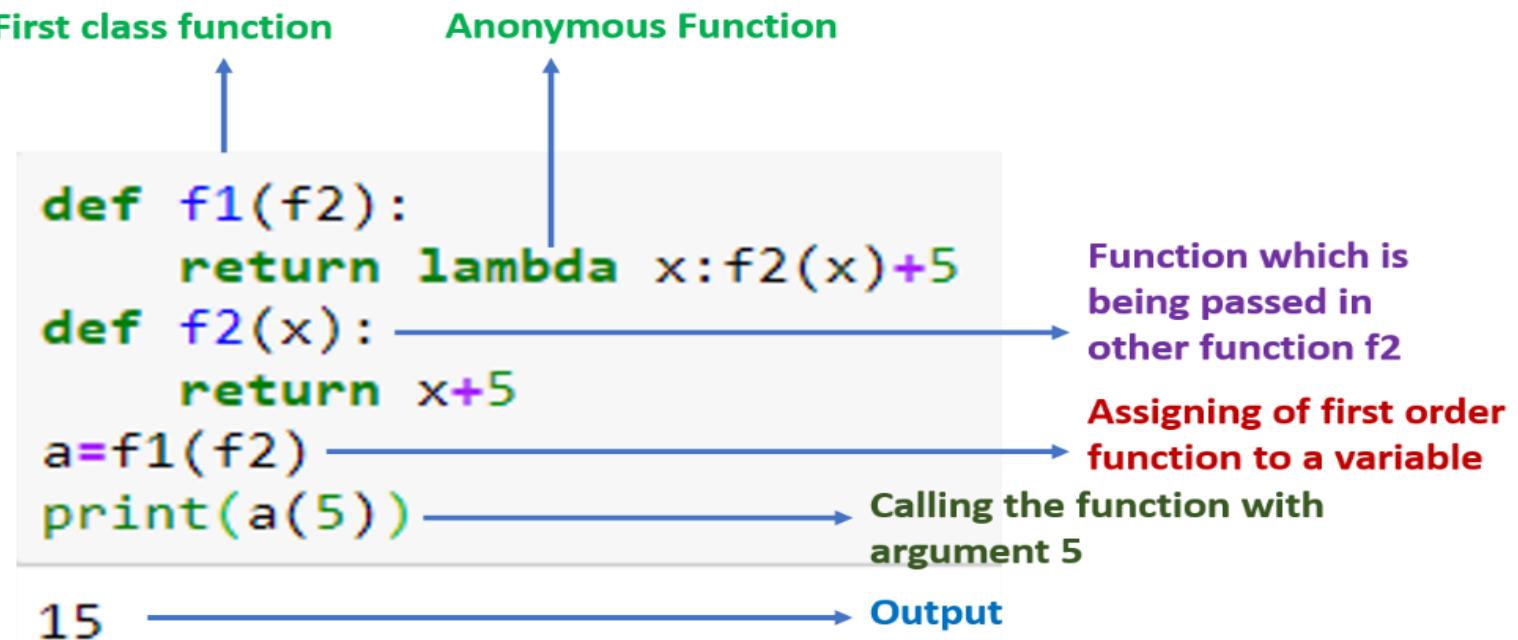
- Function can be passed as an argument into another function.
- Function can be assigned to a variable.
- Function can be returned from other function.

# Properties of High Order Function:

- Function can take other function as an argument.
- Function can return other function.

# Example 1

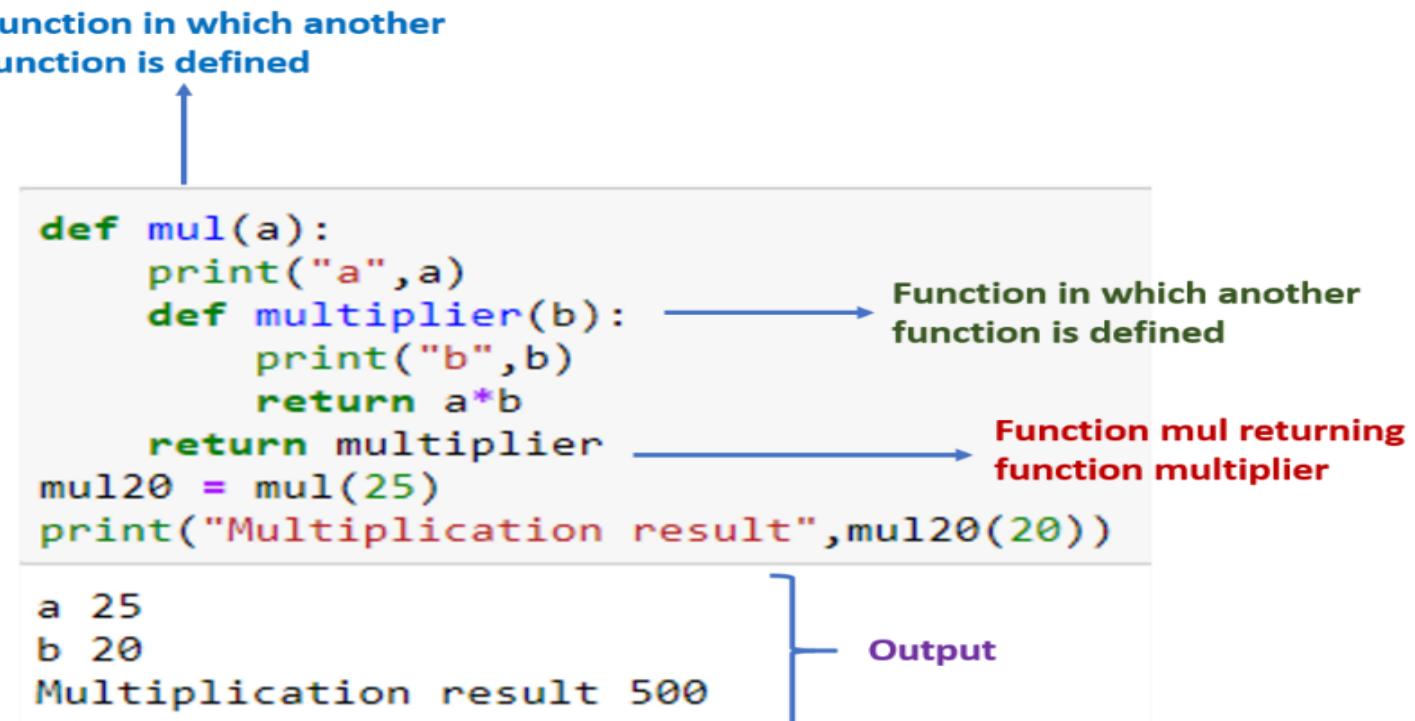
In this example function f2 is being implemented and passed as an argument in function f1. Function f1 evaluates f2 and return the final value as output.



## Example 2

A program to demonstrate to return function from a function.

In this example a function **multiplier** has been defined inside function **mul** and **mul** is returning.



# Built-in Higher order function in Python

- Built-in higher order functions are capable of taking other **function** as **argument**.
- These higher order functions are **applied over a sequence of data**.
- Here one thing to note that we can apply same functionality without using these higher order function by doing type casting and loop. But by using higher order function we avoid using of loop.

# Map ()

First higher order function we would discuss is map function.

## Syntax:

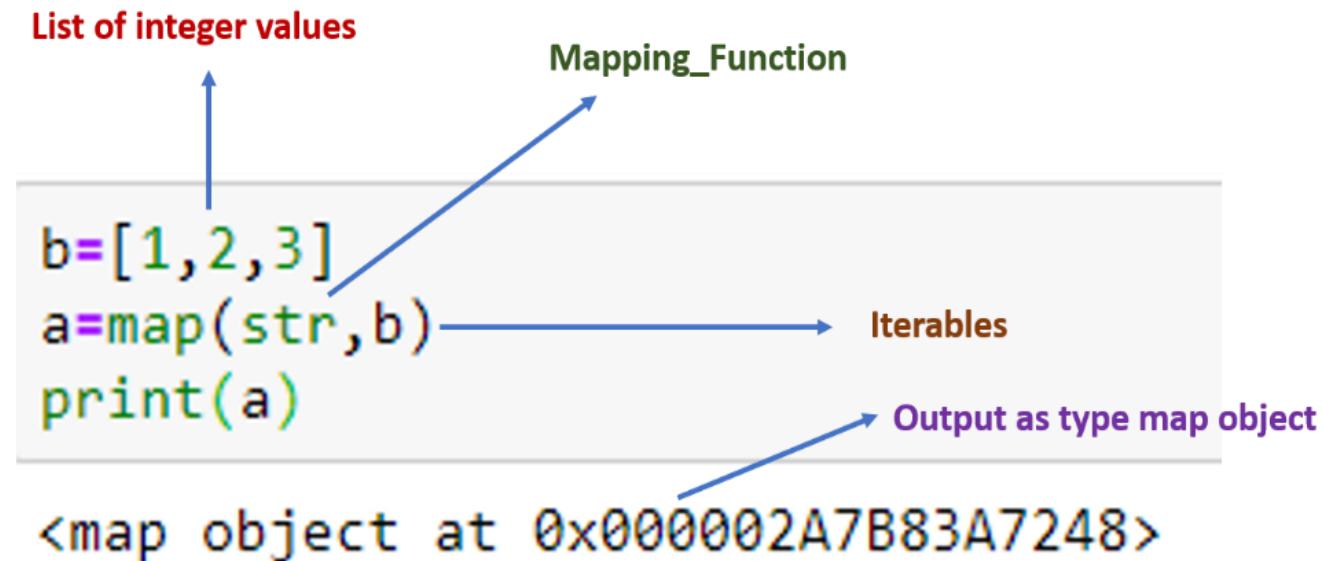
```
map(mapping_function, *iterables)
```

↑                   ↑                   ↑  
**Keyword**      **Function to be applied on values**      **Sequence of values on which mapping\_function is to be applied**

# Contd..

One important thing to note that map function always returns map object.

## Syntax:



## Contd..

To access values, we need to **type cast** map object as some sequence datatype like **lists, tuple, etc.**

```
b=[1,2,3]  
a=map(str,b)  
list(a)
```

→ Type casting of map object into list

```
[ '1', '2', '3' ]
```

→ Output List of string values

## Contd..

A program to convert a string of upper-case words into lower case.

List of Upper case words

```
str_uppercase=["ABES","ENGINEERING","COLLEGE"]
convert_map=map(str.lower,str_uppercase)
str_lowercase=list(convert_map)
print(str_lowercase)
```

Mapping\_Function

```
['abes', 'engineering', 'college']
```

# Filter()

Filter function is the high order function which takes a filtering criterion, need to apply on any sequence.

For example, in a bucket of balls of different colors, we have to find only black color balls, then filter function would work.

## Syntax:

```
filter(filtering_function, *iterables)
```

↑  
**Keyword**

↑  
**Filter to be applied on values**

↑  
**Sequence of values on which filtering\_function is to be applied**

# Contd..

A list of integers find out even numbers.

```
List of numbers
list_num=[1,2,3,4,5,6,7,8,9,10]
a=filter(lambda x: x%2 == 0,list_num)
print(a)
print(list(a))

<filter object at 0x000002A7B83A7708>
[2, 4, 6, 8, 10]
```

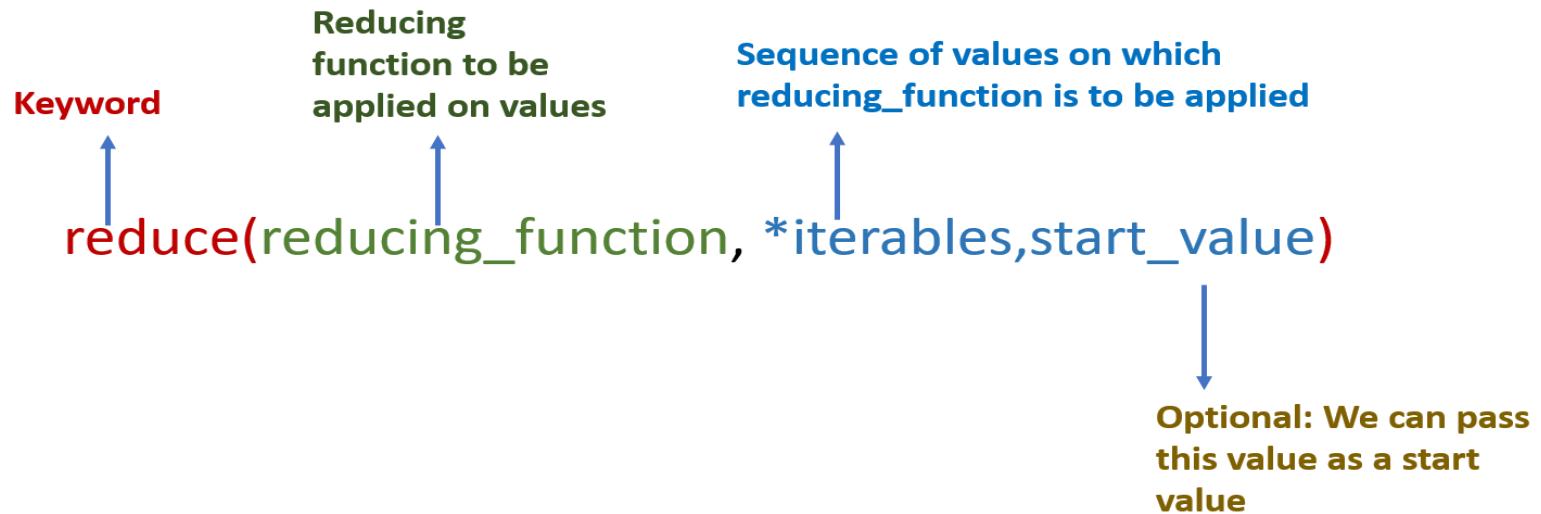
Filtering function

# Reduce()

Reduce is used in such scenarios where the values in the **iterable** would be calculated to a final value.

For example, take a collection of values (3,4,5) and we need to apply sum to it. Then we would get 12.

## Syntax:



# Example 1

A program to sum numbers of a list using reduce function.

**Importing reduce function**

**Initialized value as 15**

```
from functools import reduce
list1=[2,3,4]
sumtotal=reduce(lambda x,y:x+y,list1,15)
print(sumtotal)
```

24 —————> Output:15+(2+3+4)

# Review Questions

What will be the output of the following Python code?

```
x = ['ab', 'cd']
print(len(list(map(list, x))))
```

- a) 2
- b) 4
- c) error ←
- d) none of the mentioned

What will be the output of the following Python code?

```
x = [12, 34]
print(len(list(map(len, x))))
```

- a) 2
- b) 1
- c) error ←
- d) none of the mentioned

# Session Plan - Day 5

## 4.3 Modules

- **4.3.1 Creation of module**
- **4.3.2 Importing module**
- **4.3.3 Standard Built-In Module**

# Introduction

As length of program grows in size and to maintain the reusability of some code we start writing function. Now if you want to use same function/functions in another program instead of writing their definition then ***module comes into picture.***

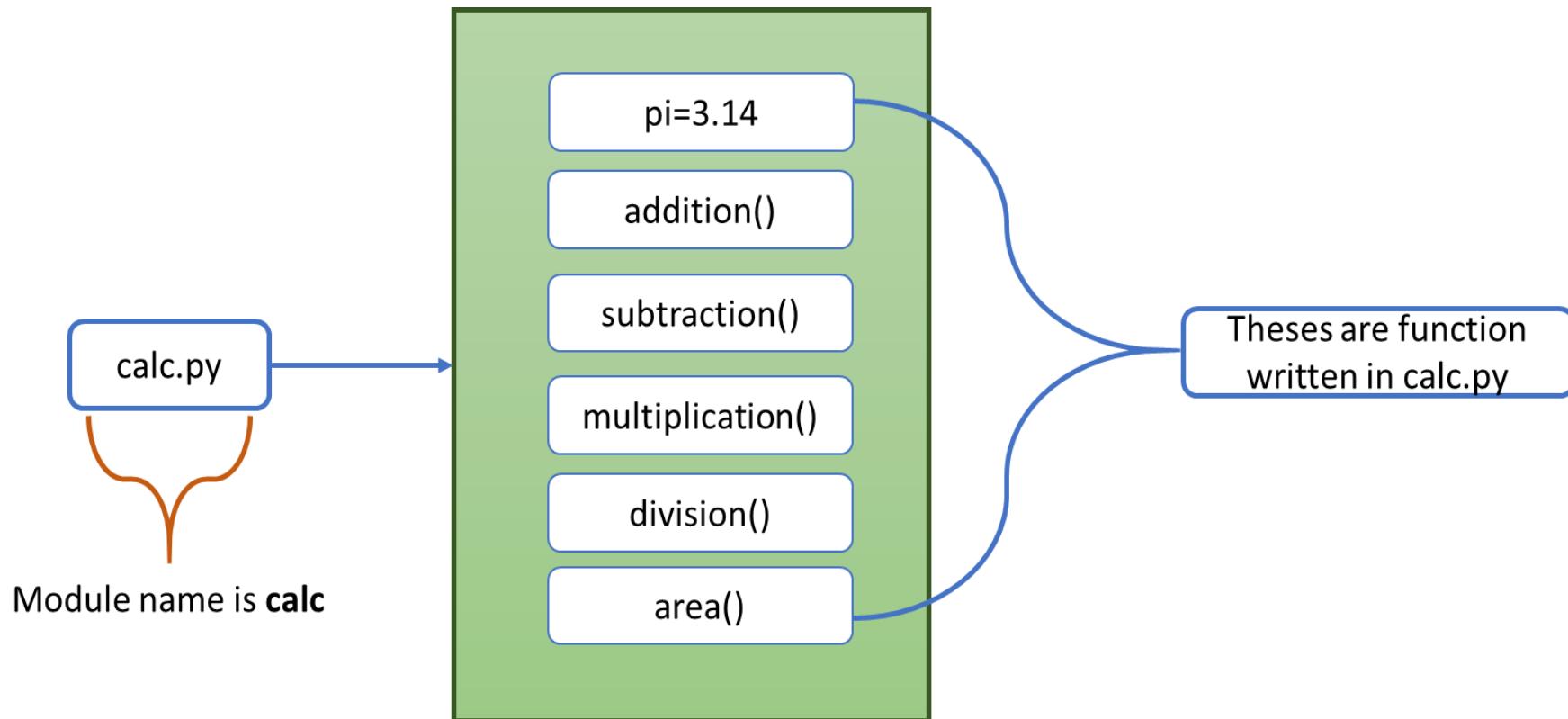
Python provides us to create our **own module** or provides use to use **rich source of given module** to increase the reusability.

Python module can be imported into any program

In python module is simply **a python file containing python code**, it may contain statements, functions, methods and classes.

# Creation of module

A file containing Python code, for example: calc.py, is called a module, and its name would be calc. A module is created simply putting all functions/statements in one python file



# Contd..

A file containing Python code, for example: calc.py, is called a module, and its name would be calc. **A module is created simply putting all functions/statements in one python file**

**Code** illustrates that how to create a module name 'calc', which contains the user defined functions of addition, subtraction, multiplication and division and area.



Whole code is saved as **calc.py**

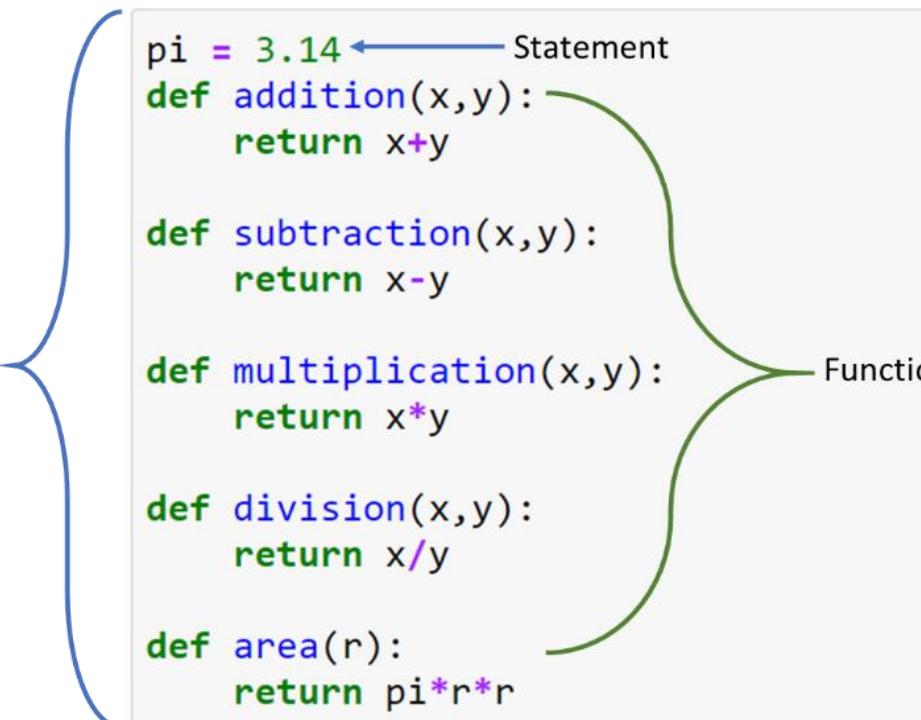
```
pi = 3.14 ← Statement
def addition(x,y):
    return x+y

def subtraction(x,y):
    return x-y

def multiplication(x,y):
    return x*y

def division(x,y):
    return x/y

def area(r):
    return pi*r**r
```



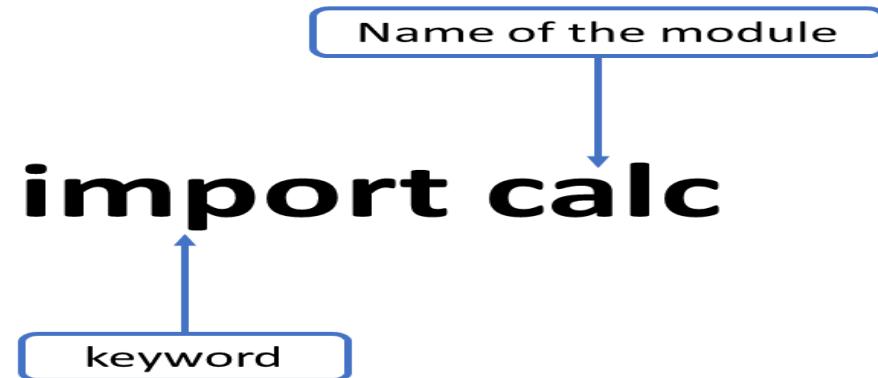
Functions definition

# Importing module

To use the module in any application or program we need to import that module using **import** keyword.

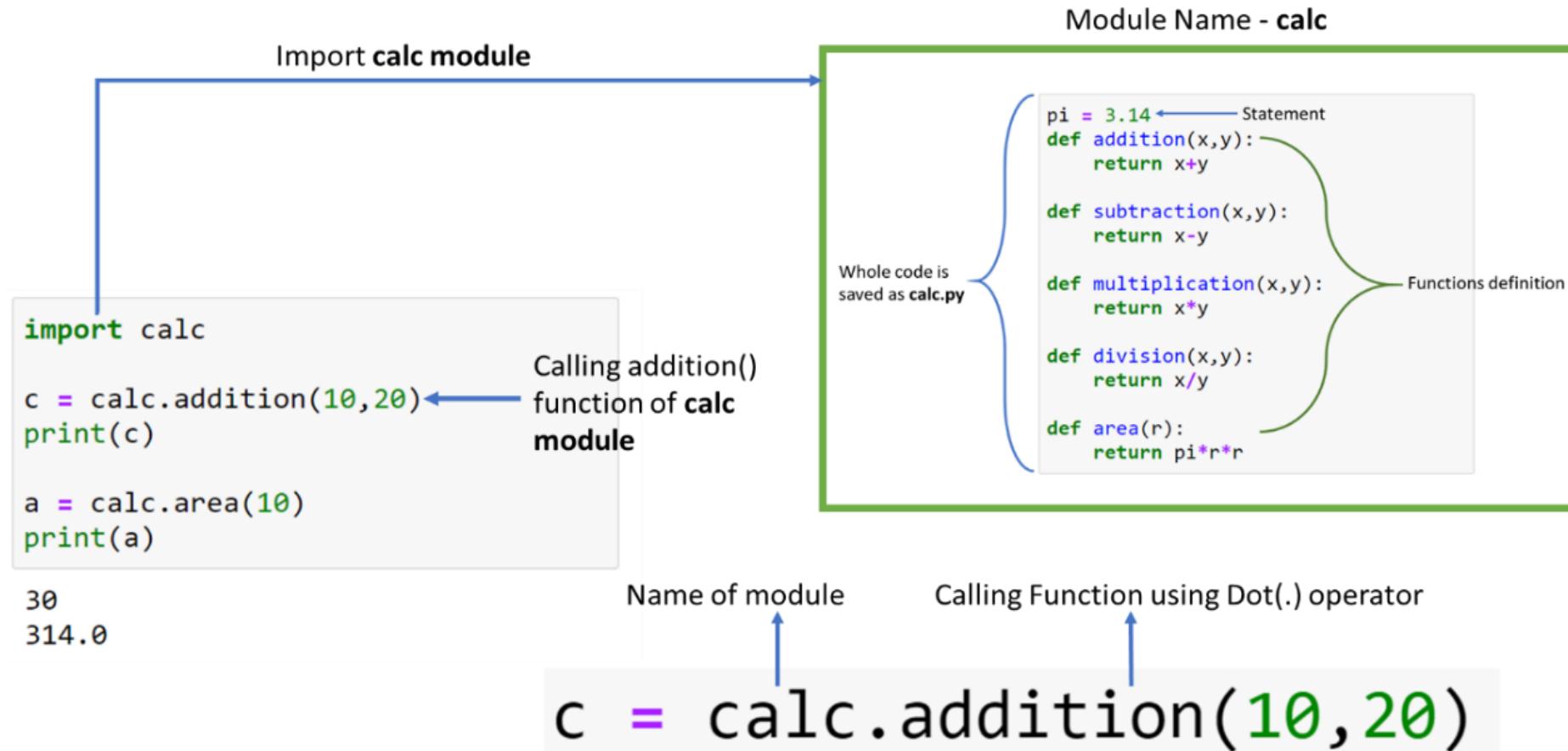
We can import module in multiple ways –

- **import** module-name
- **import ... as ...**
- **from ... import ...**



# Contd..

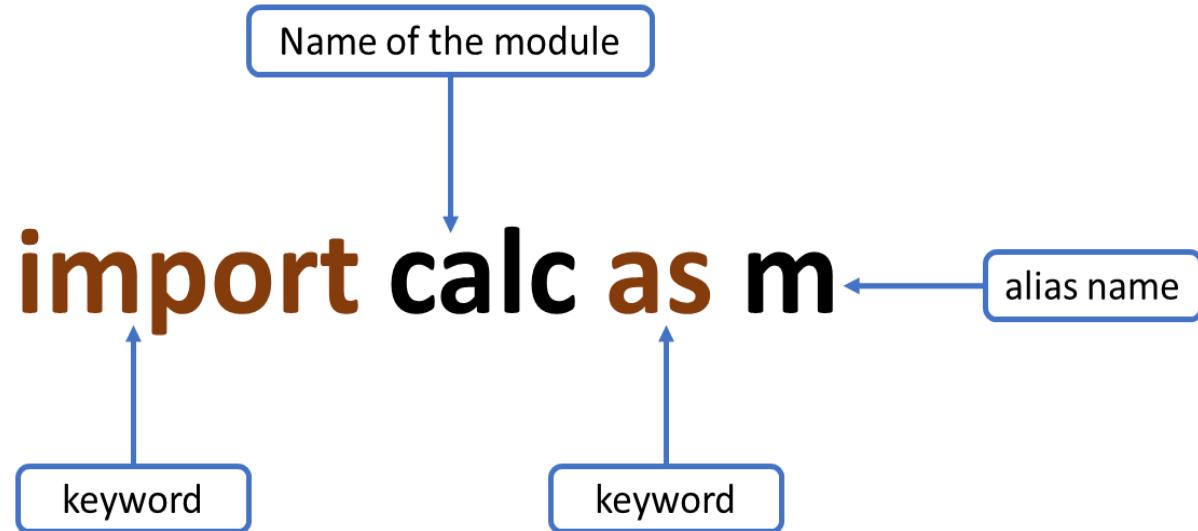
When we use “**import module-name**” syntax, it will import all functions/statements from that module and function will call using **module-name** and **dot** operator.



# Contd..

**import ... as ...**

We use **import ... as ...** to give a module name a **short alias** name while importing it.



# Contd..

**import ... as ...**

Example: We import module calc using alias name '**m**'. It will import all functions / statements from that module and function will call using alias name and **dot** operator.

```
alias name  
↓  
import calc as m  
  
c = m.addition(10,20)  
print(c)
```

30

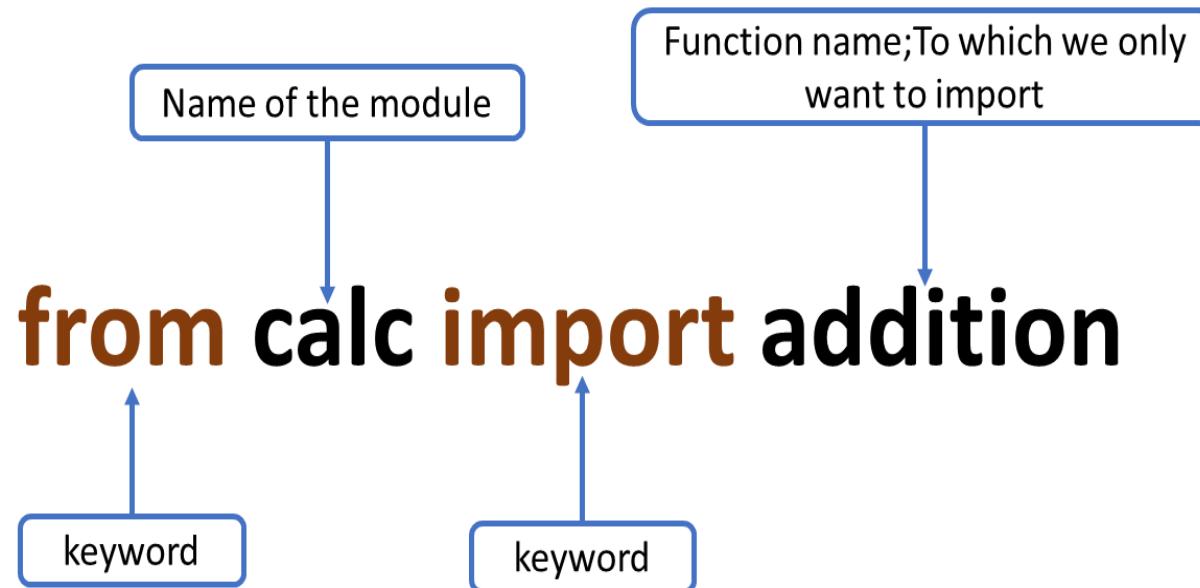
Module Name - **calc**

```
pi = 3.14  
def addition(x,y):  
    return x+y  
  
def subtraction(x,y):  
    return x-y  
  
def multiplication(x,y):  
    return x*y  
  
def division(x,y):  
    return x/y  
  
def area(r):  
    return pi*r**r
```

# Contd..

**from ... import ...**

The **from.... Import ...** statement allows us to import specific functions/variables from a module instead of importing everything.



# Contd..

**from ... import ...**

Multiple function can also be imported using comma (,) operator.

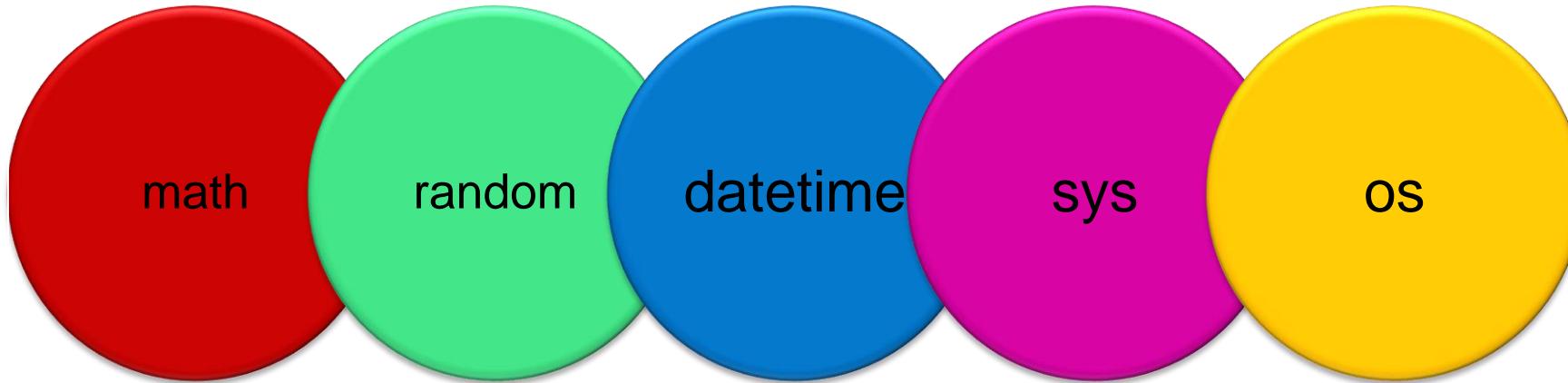
**from calc import addition, subtraction**

We can also use '\*', for importing all function/statements.

**from calc import \***

# Standard Built-In Module

Python has large number of built in functions similarly python contains some of the pre-defined modules such as:



# Contd..

## The math Module

Python has set of built-in math functions and also has an extensive math module that allows us to perform mathematical tasks on data. These include **trigonometric functions**, **representation functions**, **logarithmic functions**, **sine functions**, **cosine functions**, **exponential**, pi etc.

### math.sqrt()

This function will calculate the square root of number.

```
import math

result = math.sqrt(144)
print(result)
```

12.0

# Contd..

## math.ceil()

Rounds a number up to the nearest integer.

```
import math

result = math.ceil(14.5)
print(result)
```

15

```
import math

result = math.ceil(14.1)
print(result)
```

15

## math.exp()

Return 'E' raised to the power of different numbers

```
import math

result = math.exp(65)
print(result)
```

1.6948892444103338e+28

# Contd..

## math.factorial()

Return the factorial of a number

```
import math

result = math.factorial(6)
print(result)
```

720

## math.gcd()

Return the greatest common divisor of two integers.

```
import math

result = math.gcd(16,48)
print(result)
```

16

## math.pow()

Returns the value of x to the power of y.

```
import math

result = math.pow(2,3)
print(result)
```

8.0

# Contd..

## **math.fsum()**

Returns the sum of all items in an iterable.

```
import math

l=[1,2,3,4,5]
result = math.fsum(l)
print(result)
```

26.0

## **math.sin()**

Return the cosine of x (measured in radians).

```
import math

result = math.cos(0)
print(result)
```

1.0

Reference : <https://docs.python.org/3/library/math.html>

# Contd..

## The random Module

This module implements pseudo-random number generators for various distributions. We can generate random numbers in Python by using random module.

### random.seed()

The random number generator needs a number to start with (a seed value). Use the `seed()` method to customize the start number of the random number generator.

```
import random  
  
random.seed(10)  
print(random.random())
```

0.5714025946899135

# Contd..

## random.random()

Return random number between 0.0 and 1.0

```
import random

result = random.random()
print(result)
```

0.5780913011344704

## random.choice()

Choose a random element from a non- empty sequence like string, list, tuple etc.

```
import math

result = math.pow(2,3)
print(result)
```

8.0

## random.randint(a,b)

Return random integer between a and b, including both end points.

```
import random

result = random.randint(11,20)
print(result)
```

14

## random.shuffle()

The shuffle() method takes a sequence and reorganize the order of the items.

```
import random

color = ['Red','Black','Pink','Blue']
random.shuffle(color)
print(color)
```

['Black', 'Red', 'Blue', 'Pink']

# Contd..

## The datetime Module

In python there is no date and time data type so python date time module is used for displaying dates and times, modifying dates and time in different format.

**This module supplies classes to work with date and time. These classes provide a number of functions and built in methods to deal with dates, times and time intervals.**

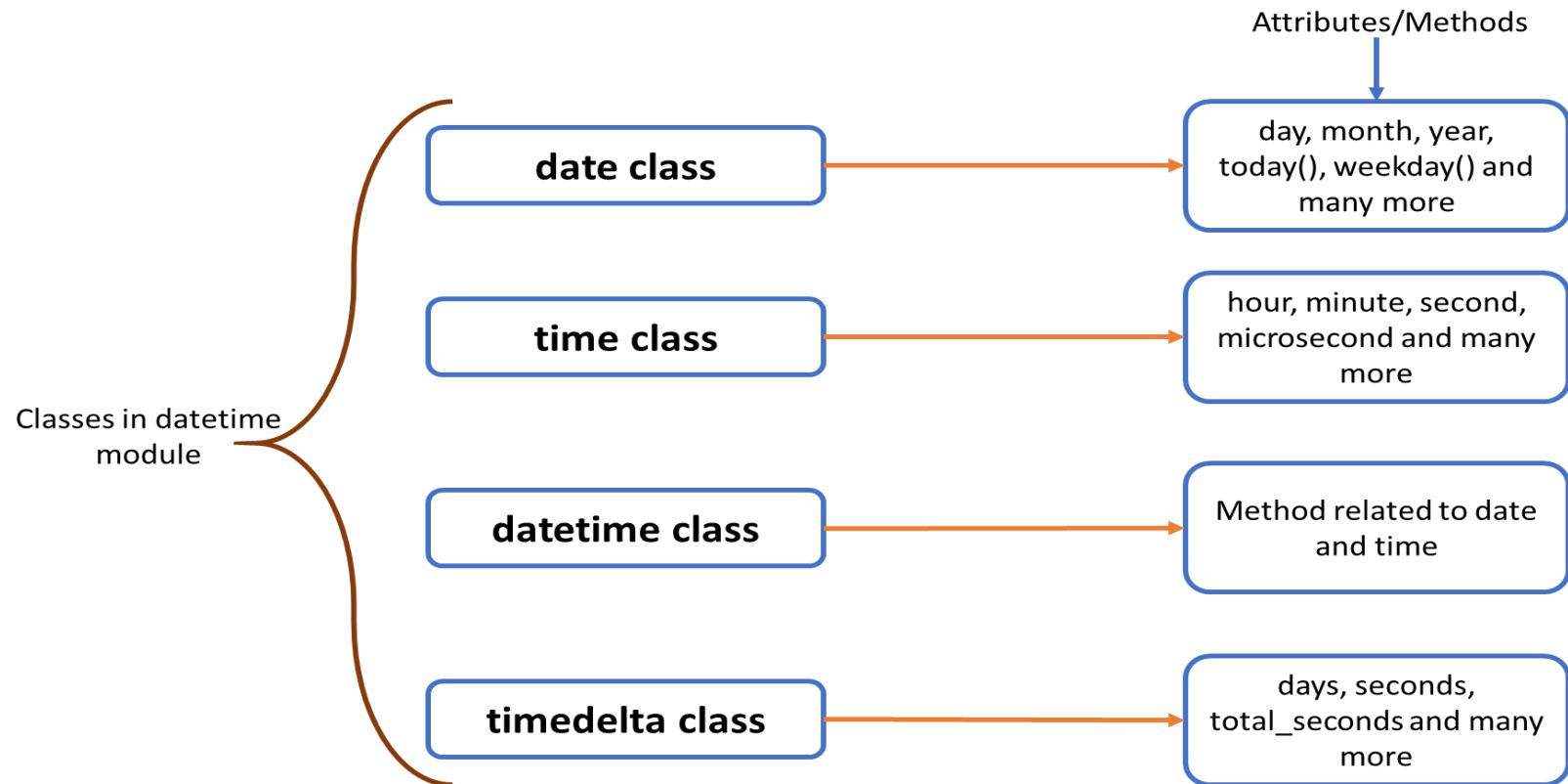
### Note:

Classes and Objects we are going to discuss in Object Oriented Programming in Python.

# Contd..

## The datetime Module

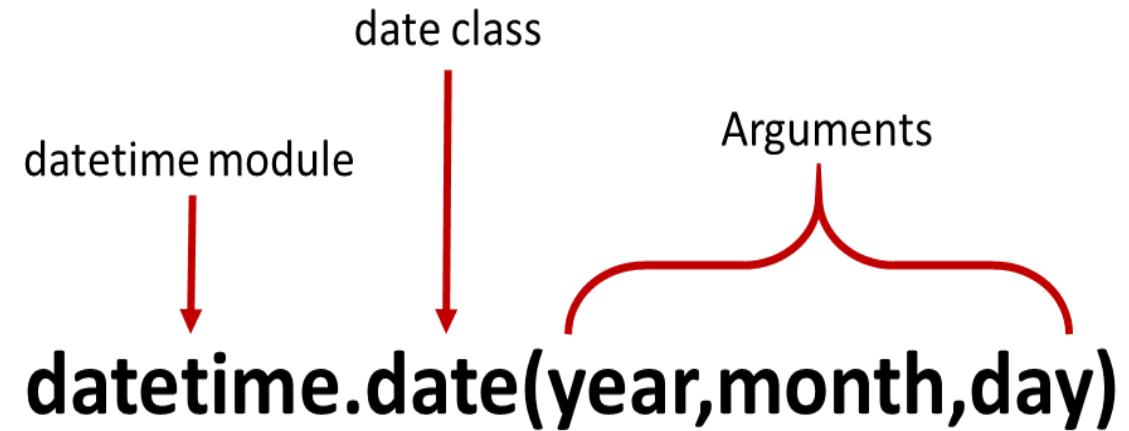
### Commonly used classes and associated methods in the datetime m



# Contd..

## The date class

A date object of date class represents a date in year, month and day.



# Contd..

All method and attributes of date object will be called using 'd' object and dot operator.

```
import datetime

d = datetime.date(2021,8,8) ← Create a date object, with year=2021, month=8, day=8
print(d)

print(f"Day = {d.day}") ← d.day give day number in date

print(f"Month = {d.month}") ← d.month give month number in date

print(f"Weekday = {d.weekday()}") ← d.weekday() return weekday. First day of week start with 0,
                                         second 1 and so on

print(f"Today's date{d.today()}") ← d.today() return current date of system
```

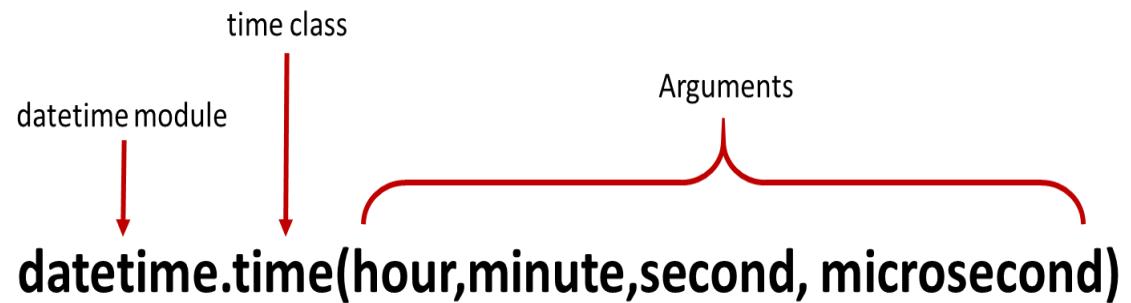
2021-08-08  
 Day = 8  
 Month = 8  
 Weekday = 6  
 Today's date 2021-08-10

Output

# Contd..

## The time class

**Time object represents local time, independent of any day.**



# Contd..

Create a time object and display hour, minute, second and microsecond in time object.

```
import datetime

t = datetime.time(9,35,12,123) ← Create a time object, with hour=9, minute=35, second=12
print(t)

print(f"Hour= {t.hour}") ← t.hour give value of hour in time

print(f"Minutes= {t.minute}") ← t.minute give value of minute in time

print(f"Microsecond= {t.microsecond}") ← t.microsecond give value of microsecond in time
```

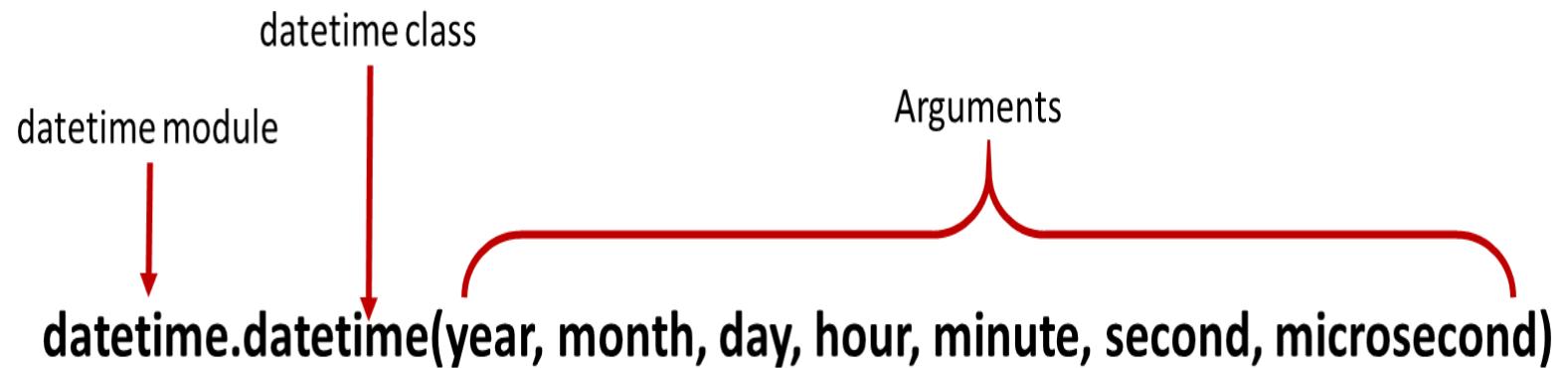
09:35:12.000123  
Hour= 9  
Minutes= 35  
Microsecond= 123

Output

# Contd..

## The datetime class

Information on both date and time is contained in this class.



# Contd..

Create a time object and display hour, minute, second and microsecond in time object.

```
import datetime

dt1 = datetime.datetime(2021, 8, 11, 10, 43, 18, 462154)
print(f"year = {dt.year}, month = {dt.month} ")
print(f"hour = {dt.hour}, minute = {dt.minute} ")
```

year = 2021, month = 8  
hour = 10, minute = 43

Create a datetime object and display year, month, hour, minute in time object.

Write a program to fetch the current datetime of system and print it

```
import datetime

dt = datetime.datetime.now()
print(dt)
```

2021-08-11 12:27:37.279229

# Contd..

## The sys Module

The sys module provides system specific **parameters and functions**. It gives us information about constants, functions, and methods of the interpreter.

Following are some important use of sys module:

- Exiting current flow of execution in Python
  - Command-line Arguments
- Getting version information of Interpreter
  - Set or get recursion depth

For more function visit - <https://docs.python.org/3/library/sys.html>)

# Contd..

## sys.version

**Gives version information of Interpreter.**

```
import sys  
print(sys.version)
```

3.8.5 (default, Sep 3 2020, 21:29:08)  
[MSC v.1916 64 bit (AMD64)]

# Contd..

## sys.argv

It returns a list of command line arguments passed to a Python script.

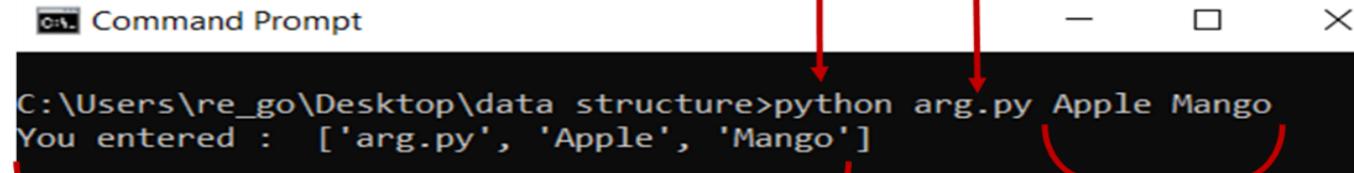


arg.py  
Python script file

```
*arg.py - C:/Users/re_go/Desktop/data structure/arg.py (3.9.1)*
File Edit Format Run Options Window Help
import sys
print('You entered : ', sys.argv)
```

Ln: 2 Col: 31

To execute Python script through command line, we use python command



Command Prompt

```
C:\Users\re_go\Desktop\data structure>python arg.py Apple Mango
You entered : ['arg.py', 'Apple', 'Mango']
```

Python script file name

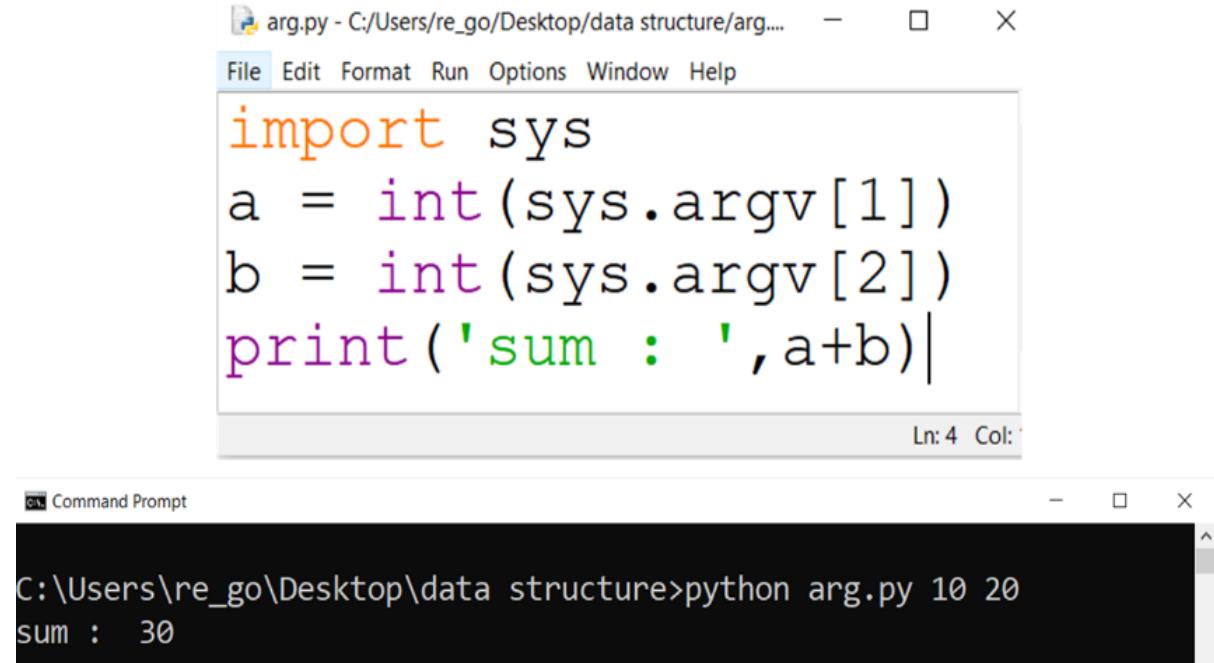
Output

Argument

# Contd..

## sys.argv

**Write a python script which print sum of given two value through command line**



The image shows a Windows desktop environment. At the top, there is a taskbar with several pinned icons. Below the taskbar, a file explorer window is open, showing a folder structure. In the foreground, there are two windows: a code editor window titled "arg.py - C:/Users/re\_go/Desktop/data structure/arg...." containing Python code, and a Command Prompt window below it. The Command Prompt window shows the output of running the script with arguments 10 and 20.

arg.py - C:/Users/re\_go/Desktop/data structure/arg....

File Edit Format Run Options Window Help

```
import sys
a = int(sys.argv[1])
b = int(sys.argv[2])
print('sum : ', a+b)
```

Ln: 4 Col: 1

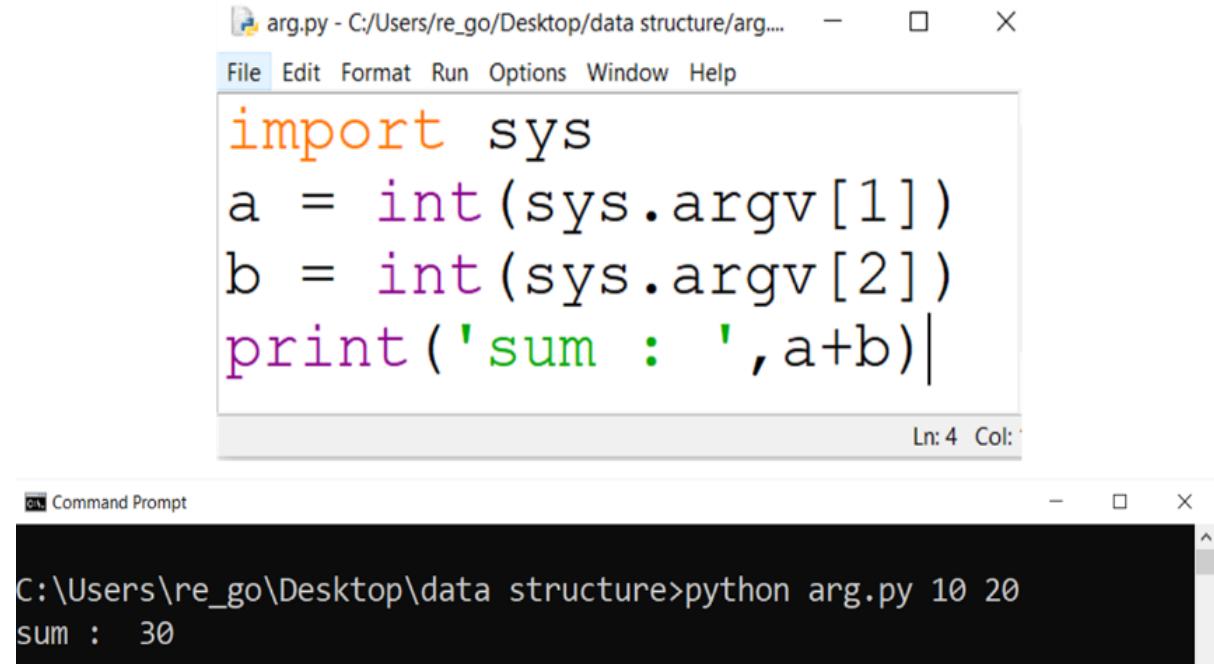
Command Prompt

```
C:\Users\re_go\Desktop\data structure>python arg.py 10 20
sum : 30
```

# Contd..

## sys.argv

**Write a python script which print sum of given two value through command line**



The image shows a Windows desktop environment. At the top, there is a taskbar with several pinned icons. Below the taskbar, there is a system tray containing a clock icon. The main area of the screen displays two windows: a code editor and a command prompt.

The code editor window is titled "arg.py - C:/Users/re\_go/Desktop/data structure/arg....". It contains the following Python code:

```
import sys
a = int(sys.argv[1])
b = int(sys.argv[2])
print('sum : ', a+b)
```

The command prompt window is titled "Command Prompt". It shows the command "python arg.py 10 20" being run, followed by the output "sum : 30".

# Contd..

## sys.getrecursionlimit() and sys.setrecursionlimit()

```
import sys  
sys.getrecursionlimit()
```

Out[85]:

3000

sys.getrecursionlimit()  
method is used to find  
the current recursion  
limit of the interpreter.

sys.setrecursionlimit()  
method is used to set  
the recursion limit of the  
interpreter

```
import sys  
sys.setrecursionlimit(100)
```

# Review Questions

- What is the output of the code shown below if the system date is 18th August, 2016?

```
tday=datetime.date.today()
```

```
print(tday.month())
```

- A. August
- B. Aug
- C. 08
- D. 8



# Review Questions

What is returned by `math.ceil(3.4)`?

- a) 3
- b) 4
- c) 4.0
- d) 3.0

What is returned by `math.ceil(3.4)`?

- a) 3
- b) 4
- c) 4.0
- d) 3.0

# Review Questions

➤ What will be the output of the following Python code?

```
import time
```

```
print(time.strftime("%d-%m-%Y"))
```

- a) Print today's date in string format dd-mm-yy
- b) Print today's date in string format dd-mm-yyyy
- c) Error
- d) None

➤ State whether true or false.

```
t1 = time.time()
```

```
t2 = time.time()
```

```
t1 == t2
```

- a) True

- b) False

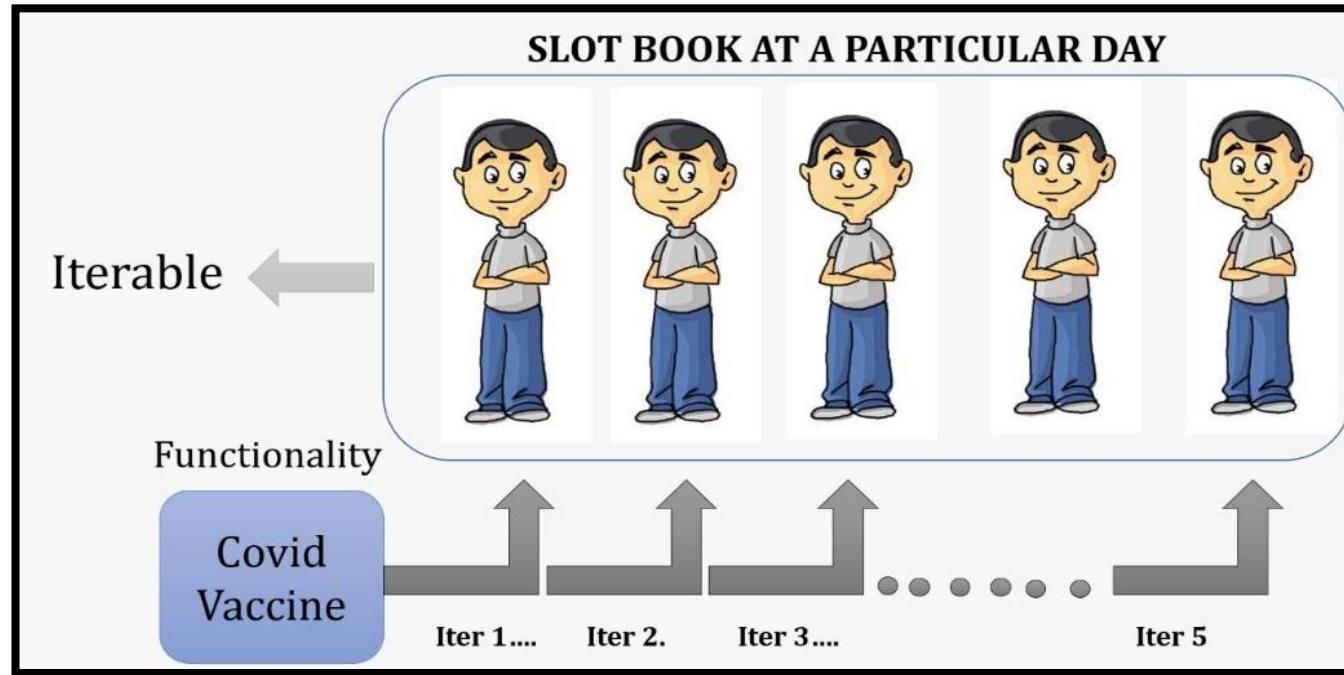
# Session Plan - Day 6

## 4.3 Modules

- **4.4 Iterative Built-in Functions**
  - Enumerate**
  - Zip**
  - Sorted**
  - Zip**

# Iterative Built-in Functions

**Enumerate:** This is built-in function provided by Python. This function assigns count values to the iterable



## What is iterable?

**Iterable** in python is a collection of items, sequences like lists, tuple or any object which have multiple values on which a repeat operation can be performed, like

```
name = ['amit', 'ankur', 'aditya', 'anshul',
'anmol'],
```

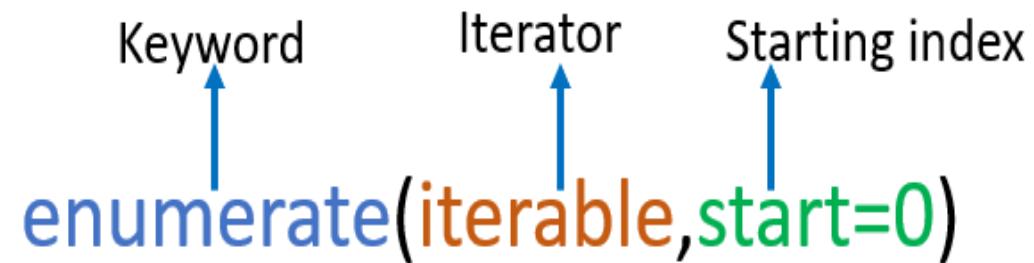
here the **name** has five items in it.

## Contd..

When using the **iterators**, we need to keep track of the number of items in the iterator. This is achieved by built in method called enumerate.

This enumerated object can directly be used for loops or converted into a list of tuples using **list ()** method.

**Note: by default enumerate starts from 0 but we can change this start value as per our convenience.**



The diagram illustrates the components of the `enumerate` function. It shows the function name `enumerate` in blue, followed by two arguments in parentheses: `(iterable, start=0)`. Three blue arrows point upwards from the text labels to their corresponding parts in the function call:

- A vertical arrow labeled "Keyword" points to the word `enumerate`.
- A vertical arrow labeled "Iterator" points to the word `iterable`.
- A vertical arrow labeled "Starting index" points to the parameter `start=0`.

```
enumerate(iterable, start=0)
```

Keyword      Iterator      Starting index

# Contd..

**Example : Demonstrating the enumerate usage in printing days of week.**

```
days= { 'Mon', 'Tue', 'Wed', 'Thu'} → Set
enum_days = enumerate(days)
print(type(enum_days)) → Print the days of the week
# converting it to a list
print(list(enum_days))

# changing the default counter to 5
enum_days = enumerate(days, 5) → Print the days of the week
print(list(enum_days))

<class 'enumerate'>
[(0, 'Mon'), (1, 'Wed'), (2, 'Thu'), (3, 'Tue')]
[(5, 'Mon'), (6, 'Wed'), (7, 'Thu'), (8, 'Tue')]
```

Output

# Contd..

## Enumerate with for loop

Syntax

for i value in enumerate(collection):  
 keyword    Sequence

### Example

```
k = ['ABES', 'Engineering', 'College', 'Ghaziabad'] → list
for i in enumerate(k): → For loop
    print(i)
```

(0, 'ABES')  
(1, 'Engineering')  
(2, 'College')  
(3, 'Ghaziabad')

Output

# Contd..

## Zip

The function returns a zip object, which is an iterator of tuples where the **first item** in each passed iterator is paired together, and then the **second item** in each passed iterator are paired together etc.



### Example

Keyword                    Iterator  
`zip(iterator1,iterator2,iterator3)`

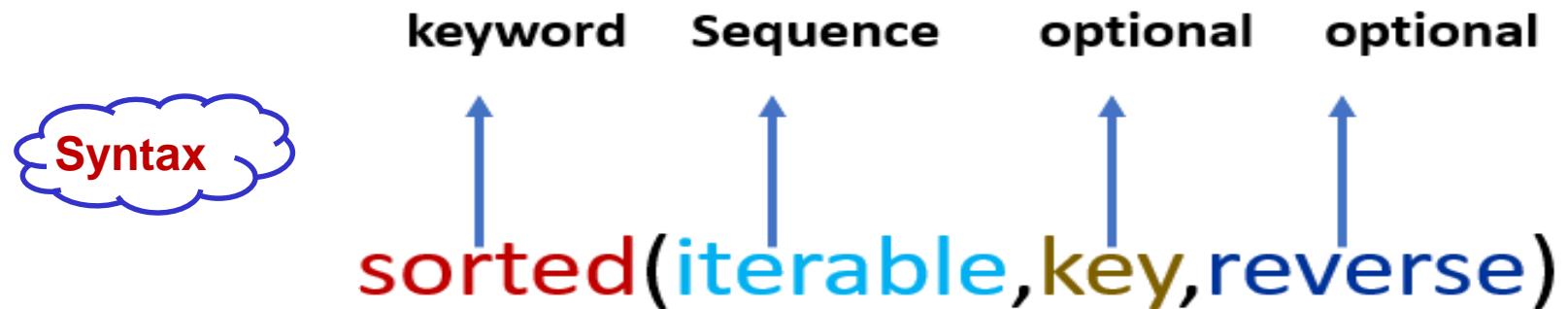
```
a = ("John", "Charles", "Mike") → tuple
b = ("Jenny", "Christy", "Monica", "Vicky") → tuple
x = zip(a, b)
print(list(x))
```

`[('John', 'Jenny'), ('Charles', 'Christy'), ('Mike', 'Monica')]` } Output

# Contd..

## Sorted

Python contains a special built in method for **sorting** the data in a collection. It preserves the order of previous sequence or collection.



# Contd..

## Sorted Example

```
x = [2, 8, 1, 4, 6, 3, 7] → List
# x is a list
print ("Sorted List returned :"),
print (sorted(x))
# this will print the List in ascending order
print ("Reverse sort :"),
print (sorted(x, reverse = True)) → This will print in reverse order
#this will print the List in descending order if reverse is set true
print ("Original list not modified :"),
print (x)
```

```
Sorted List returned :
[1, 2, 3, 4, 6, 7, 8]
Reverse sort :
[8, 7, 6, 4, 3, 2, 1]
Original list not modified :
[2, 8, 1, 4, 6, 3, 7]
```

Output

# Contd..

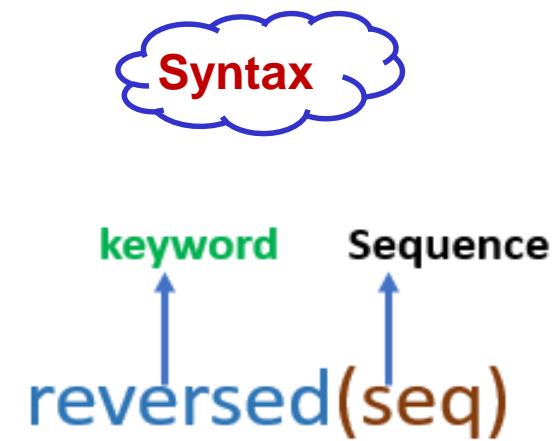
## Reversed

Reversed () function returns the iterator in a reverse order. Iterator can be any sequence such as list, tuple etc.

### Example

```
# for string
s = 'YELLOW' → String
print(list(reversed(s)))
# for tuple
t= ('P', 'y', 't', 'h', 'o', 'n') → Tuple
print(list(reversed(t)))
# for range
r = range(5, 9) → range
print(list(reversed(r)))
# for List
l = [1, 2, 4, 3, 5]
print(list(reversed(l)))
```

[ 'W', 'O', 'L', 'L', 'E', 'Y']  
[ 'n', 'o', 'h', 't', 'y', 'P']  
[ 8, 7, 6, 5]  
[ 5, 3, 4, 2, 1]



# Review Questions

- Which of the following functions can help us to find the version of python that we are currently working on?
  - a) sys.version
  - b) sys.version()
  - c) sys.version(0)
  - d) sys.version(1)
  
- Suppose there is a list such that: l=[2,3,4]. If we want to print this list in reverse order, which of the following methods should be used?
  - a) reverse(l)
  - b) list(reverse[(l)])
  - c) reversed(l)
  - d) list(reversed(l))

# Review Questions

- Which of the following functions is not defined under the sys module?
  - a) sys.platform
  - b) sys.path
  - c) sys.readline
  - d) sys.argv
  
- To obtain a list of all the functions defined under sys module, which of the following functions can be used?
  - a) print(sys)
  - b) print(dir.sys)
  - c) print(dir[sys])
  - d) print(dir(sys))

# Summary

- In this unit you learnt the importance of using functions and how to use functions.
- Further you learnt how to define functions, to write functions with different argument types.
- Further you learnt describe the difference between the in-built functions and user defined functions,
- Further you to write user defined functions, to use Functions in-built functions, import math module in to a Python program and to use lambda functions.

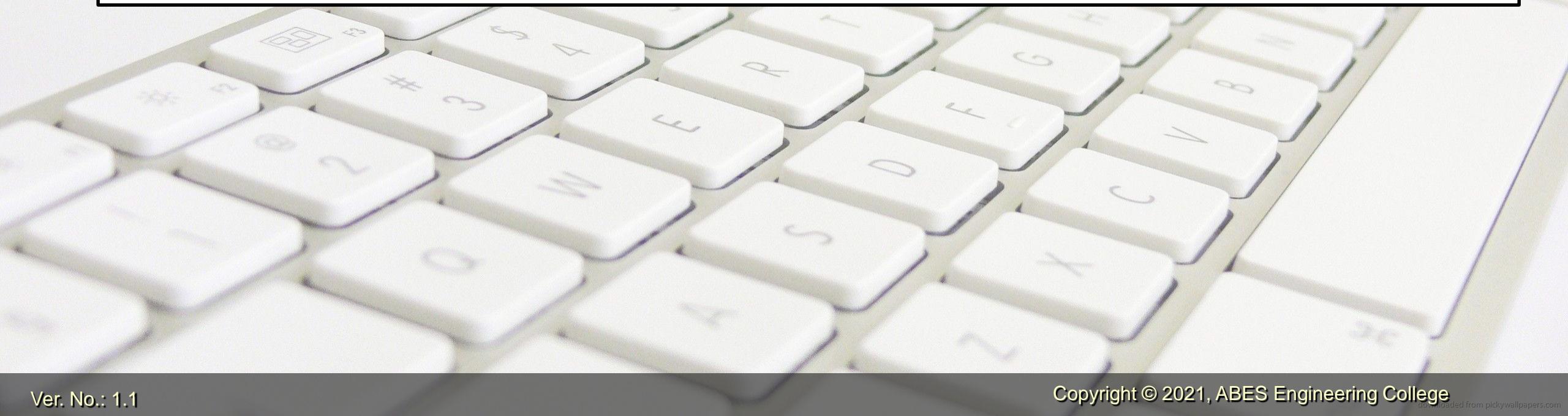
# References

1. <https://docs.python.org/3/tutorial/controlflow.html>
2. Think Python: An Introduction to Software Design, Book by Allen B. Downey
3. Head First Python, 2nd Edition, by Paul Barry
4. Python Basics: A Practical Introduction to Python, by David Amos, Dan Bader, Joanna Jablonski, Fletcher Heisler
5. <https://www.fullstackpython.com/turbogears.html>
6. <https://www.cubicweb.org>
7. <https://pypi.org/project/Pylons/>
8. <https://www.upgrad.com/blog/python-applications-in-real-world/>
9. <https://www.codementor.io/@edwardbailey/coding-vs-programming-what-s-the-difference-yr0aeug9o>

# Thank You

---

# 5. Regular Expression



# General Guideline

© (2021) ABES Engineering College.

This document contains valuable confidential and proprietary information of ABESEC. Such confidential and proprietary information includes, amongst others, proprietary intellectual property which can be legally protected and commercialized. Such information is furnished herein for training purposes only. Except with the express prior written permission of ABESEC, this document and the information contained herein may not be published, disclosed, or used for any other purpose.

# Topics Covered

## Day 1

5.1 GETTING FAMILIAR  
WITH REGULAR  
EXPRESSION

5.1.1  
INTRODUCTION TO  
REGEX  
5.1.2 REGULAR  
EXPRESSION IN  
PYTHON.

## Day 2

5.2 HOW TO  
CREATE PATTERN

## Day 3

5.3 REGEX  
ESSENTIAL  
METHODS

5.3.1 SEARCH ()  
METHOD  
5.3.2 MATCH ()  
METHOD  
5.3.3 FINDALL ()  
METHOD  
5.3.4 FLAG  
ARGUMENT IN  
REGEX METHOD

# Session Plan - Day 1

## 5.1 GETTING FAMILIAR WITH REGULAR EXPRESSION

### 5.1.1 INTRODUCTION TO REGEX

### 5.1.2 REGULAR EXPRESSION IN PYTHON.

# Introduction

# Regular Expression

# RegEx

# Without Regex

➤ I want that each student password should:

- Start with letter and end with a number
- It should be of length 8 and
- It should contain at least 1 special character.

➤ Solutions using what we have learnt:

- Check each condition using string matching one by one.
- Compare length of each password using len() function
- Specifically check for first and last position using membership

**Text from which pattern to be matched**



**Pattern to match**

- String matching
- Compare Length
- Using Membership

# Introduction to Regex

- Regular expression is also known as regexes or regex.
- A regex is a character series that specifies a pattern for string matching.
- Text can be searched, edited, and manipulated using regex.



# Raw Python strings

- Raw strings are raw string literals that treat backslash (\ ) as a literal character.
- It is useful when we want to have a string that contains backslash(\) and don't want it to be treated as an escape character.
- For example, if we try to print a string with a “\n” inside, it will add one line break.
- But if we mark it as a raw string, it will simply print out the “\n” as a normal character.
- Python raw strings are prefixed with ‘r’ or ‘R’. Prefix a string with ‘R’ or ‘r’ and it will be treated as a raw string.

```
raw_s = r'Hi\nHello'  
print(raw_s)
```

# Raw Python strings

- Raw strings are raw string literals that treat backslash (\ ) as a literal character.
- It is useful when we want to have a string that contains backslash(\) and don't want it to be treated as an escape character.
- For example, if we try to print a string with a “\n” inside, it will add one line break.
- But if we mark it as a raw string, it will simply print out the “\n” as a normal character.
- Python raw strings are prefixed with ‘r’ or ‘R’. Prefix a string with ‘R’ or ‘r’ and it will be treated as a raw string.

# Raw Python strings

```
text= "This is a \n normal string"
print(text)

raw_text = r"This is a \n raw string"
print(raw_text)
```

'r' treats \ as a  
literal character

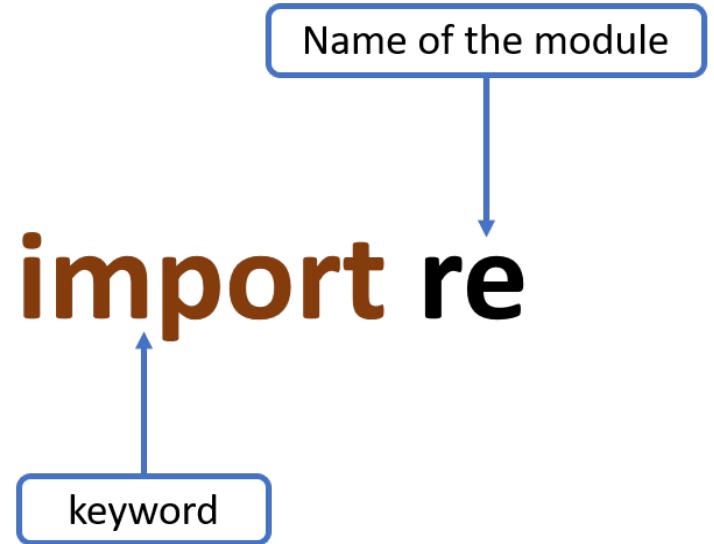
This is a  
normal string  
This is a \n raw string

} Output

- we can see that the first string text includes one new-line and the second raw string also include one new-line character.
- But the new line was printed as '\n' for the second string.

# Regular Expression in Python

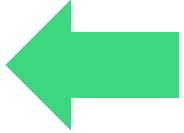
- To use Regex in python program we need to import a module re
- The re module in Python contains the regex functions.
- Many useful functions and methods are available in the re module.



# Can you answer these questions?

1. Which module in Python supports regular expressions?

A) **re**



B) **regex**

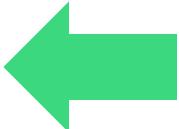
C) **reregex**

D) **piregex**

# Can you answer these questions?

2. Using regex we can

- A) Search
- B) Edit
- C) Modify
- D) All of the above



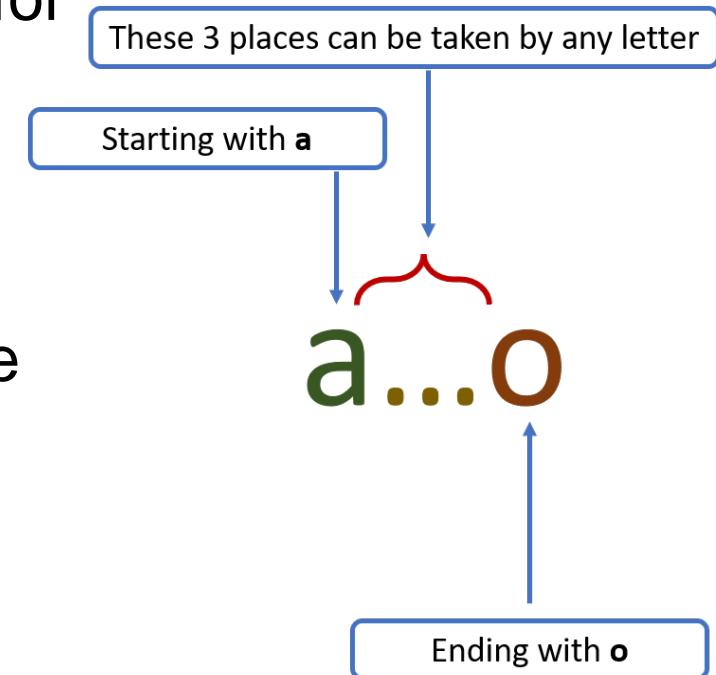
# Session Plan - Day 2

## 5.2 HOW TO CREATE PATTERN

### Meta Characters

# How to create pattern

- Let's understand suppose I want to search my text book for a word which start with letter a and ending with o and having length of 5.
- Now, if we have understood that in this example, we have specified a pattern and words like 'aggro','amino','anglo' match our pattern.



# Meta Characters

Now real **question** is how we would define a pattern in a python program.

To **answer** this, we have

# Meta Characters.

# Meta Characters

- Meta characters are characters which contain a special meaning when they are read in regular expression environment

[ ] . ^ \$ \* + ? { } ( ) \ |

# Meta Characters

- Square Brackets ([ ])
- Whatever character we would write inside square bracket, that character would be matched.

Pattern	String	Matched or Not
[abcd]	a	1 match
[abcd]	b	1 match
[abcd]	c	1 match
[abcd]	d	1 match
[abcd]	ab	2 match
[abcd]	abd	3 match
[abcd]	ab cd ef	4 match
[abcd]	efgh i	No match

# Meta Characters

- We can also define interval inside square bracket like [a-d] is same as [abcd], [0-9] is same as [0123456789].
- If we put ^ (caret symbol) as a prefix of pattern inside square bracket, then it reverses the meaning of pattern.
- For example:
  - [^de] means any character except d and e.
  - [^1-3] means any number except 1,2 and 3.

# Meta Characters

- Period or dot(.)
- Period matches single character in the string. In the given example there are two dots it means that it would match 2 consecutive characters.

Pattern	String	Matched or Not
..	a	No match
..	b	No match
..	c	No match
..	d	No match
..	ab	1 match
..	abde	2 match

# Meta Characters

- Carot (^)
- If we want to check whether string is starting with a mentioned character.
- For example, if I want to check if a string starting with letter Z, then pattern I would create is “^Z”.

Pattern	String	Matched or Not
$^a$	a	1 match
	b	No match
	ab	1 match
	abc	1 match
	ab	1 match
	abde	1 match
$^cd$	c	No match
	d	No match
	cd	1 match
	ced	No match

# Meta Characters

- Dollar Sign (\$)
- If we have to check the ending character of string then we use dollar.
- For example,

Pattern	String	Matched or Not
b\$	b	1 match
	cab	1 match
	ball	No match

# Meta Characters

- Star(\*)
- Star check if string has 0 or more occurrence.
- For example,

Pattern	String	Matched or Not
am*b	ab	1 match
	amb	1 match
	ammb	1 match
	amin	No match

# Meta Characters

- Plus(+)
- Plus check if string has 1 or more occurrence.
- For example,

Pattern	String	Matched or Not
a+b	b	No match
a+b	ab	1 match
a+b	aab	1 match

# Meta Characters

- Question Mark( ?)
- Question mark check if string has 0 or 1 occurrence.
- For example,

Pattern	String	Matched or Not
a <b>?</b> b	b	1 match
	ab	1 match
	aba	1 match

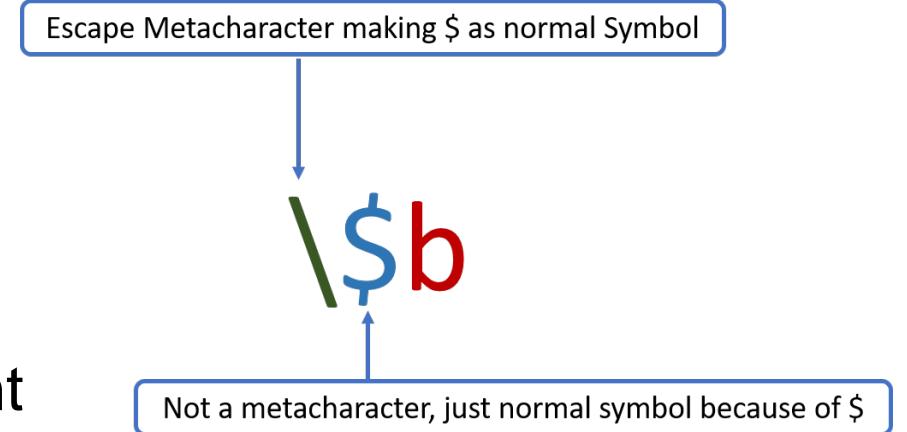
# Meta Characters

- Braces({})
- Braces are used when we want to specify at least and at most repetitions
- For example  $a\{2,5\}$  means at least 2 occurrences of a and maximum 5 occurrences of a.

Pattern	String	Matched or Not
$a\{2,5\}$	a	No match
	aa	1 match
	abca	No match
	aaaaabaa	2 match

# Meta Characters

- Backslash(\)
- We can have some requirements where we want to use metacharacter symbol in their normal form,
- It means we want regular expression environment to treat them as a symbol not as a metacharacter.
- In this example \$ is dollar not metacharacter.



# Meta Characters

- Alternation(|)
- It is used as OR operator for example,

It will search for either a and b in the string

a | b

# Meta Characters

- Group(())
  - If we want to join more than one patterns then group metacharacter can be used.
  - For example, if I want my string should start with either a or m followed by numeric
1. Pattern created is as shown in figure.

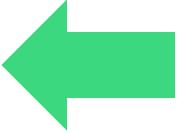
Pattern	String	Matched or Not
(a m)1	a1	1 match
	b1	No match
	m1	1 match
	am1	1 match

# Can you answer these questions?

2. Which character stand for Starts with in regex?

A) \$

B) ^



C) &

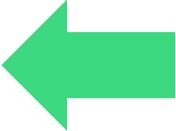
D) #

# Can you answer these questions?

2. Which character stand for Zero or more occurrences in regex?

A) #

B) @

C) \* 

D) |

# Session Plan - Day 3

## 5.3 REGEX ESSENTIAL METHODS

**5.3.1 SEARCH () METHOD**

**5.3.2 MATCH () METHOD**

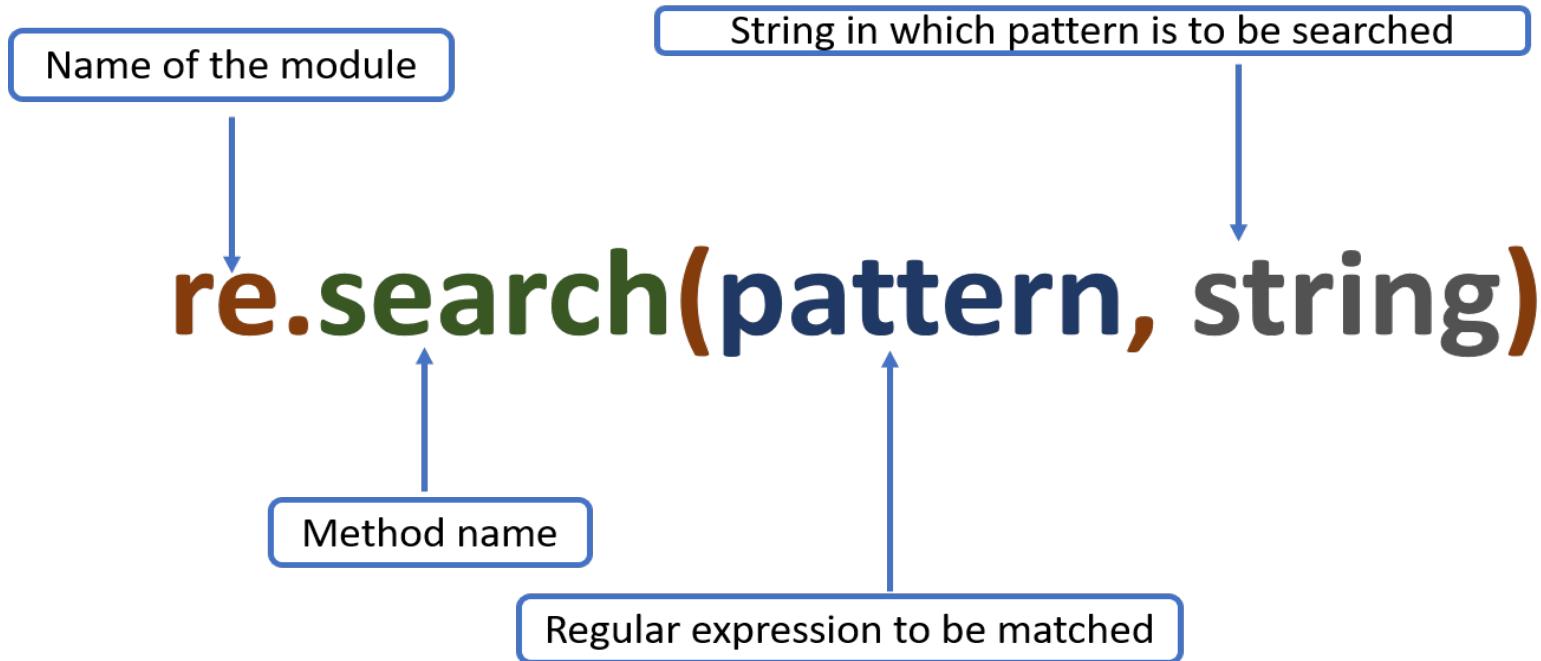
**5.3.3 FINDALL () METHOD**

**5.3.4 FLAG ARGUMENT IN REGEX METHOD**

# RegEx essential methods

- Now we understand how pattern can be created using metacharacters.
- Re module has many useful functions.
- These functions help us to search and match specified pattern.

# Search () method



# Example

```

Module Name
↓
import re
string="ABES Engineering College"
ma=re.search("AB",string)
print(ma)
if(ma):
    print("Search Successful")
else:
    print("Search UnSuccessful")

<re.Match object; span=(0, 2), match='AB'>
Search Successful

```

**String where pattern is to be searched**

**Search method in which we are giving pattern AB to be searched in string**

**Checking if Search is successful or not**

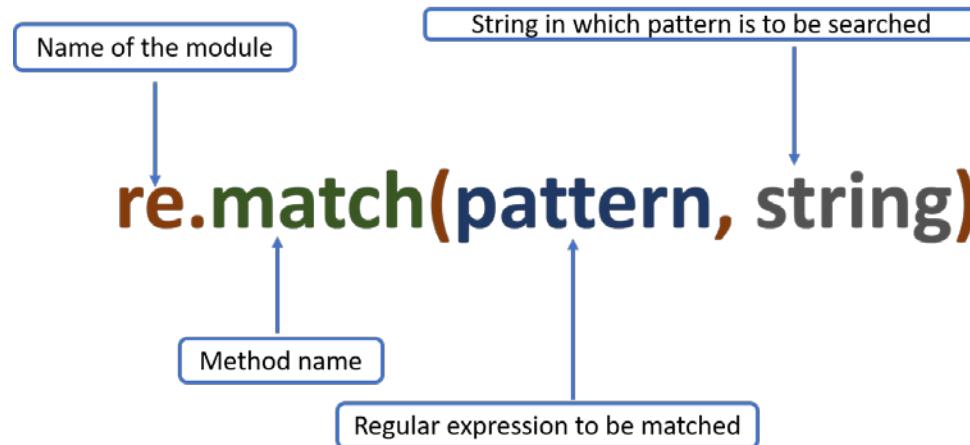
**Output**

**Index at which match is found**

**Length of pattern which is 2 in this case**

# Match () method

- Match works same as search method
- But difference is match method only look for the pattern in the beginning while search scans complete string.



# Example

String in which pattern to be searched  
**Test is at the end**

```
import re
string="String sample Test"
ma=re.match("Test",string)
print(ma)
if(ma):
    print("Search Successful")
else:
    print("Search UnSuccessful")
```

None  
Search UnSuccessful }      Output

String in which pattern to be searched  
**Test is at the Beginning**

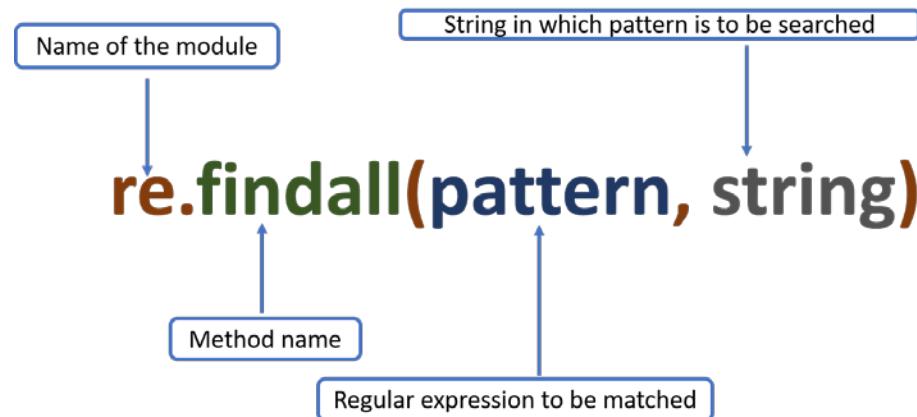
```
import re
string="Test String sample"
ma=re.match("Test",string)
print(ma)
if(ma):
    print("Search Successful")
else:
    print("Search UnSuccessful")
```

Search  
Successful  
because  
pattern is at  
the beginning

{<re.Match object; span=(0, 4), match='Test'>  
Search Successful

# Findall () method

- We have seen that search and match method find single occurrence in the string.
- The findall () method is used to find "all" instances of a given pattern in the string.



# Example

String in which pattern to be searched  
**Test** is coming two times



```
import re
string="Test String sample Test"
ma=re.findall("Test",string)
print(ma)
if(ma):
    print("Search Successful")
else:
    print("Search UnSuccessful")
```

[ 'Test', 'Test' ] }  
Search Successful }

Output

# Flag Argument in Regex Method

- Flag is the optional argument in regex methods like search, match and findall.
- Flag is used when we need to modify the standard behavior of regex patterns.

# Flag Argument in Regex Method

- For example, lets understand a scenario if I am looking for the occurrences of character “i” in the string and I don’t want to consider the **upper** and **lower** case as a different character so small case i and capital case I both are same as per requirement then I can set this flag as ignore case.

Flag Set to ignore case

```
import re
pattern="i"
string1="Information is immediate"
s1=re.findall(pattern,string1,flags=re.IGNORECASE)
s1
```

[ 'I', 'i', 'i', 'i', 'i' ] } Contains small case i and capital case I

# Flag Argument in Regex Method

- There are various options for flags as per following table.

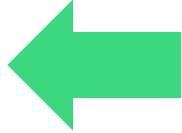
Syntax	Long syntax	Meaning
re.I	re.IGNORECASE	Ignore case.
re.M	re.MULTILINE	Make begin/end {^, \$} consider each line.
re.S	re.DOTALL	Make . match newline too.
re.U	re.UNICODE	Make {\w, \W, \b, \B} follow Unicode rules.
re.L	re.LOCALE	Make {\w, \W, \b, \B} follow locale.
re.X	re.VERBOSE	Allow comment in regex.

# Can you answer these questions?

2. The expression `a{5}` will match \_\_\_\_\_ characters with the previous regular expression.

A) 5 or less

B) exactly 5



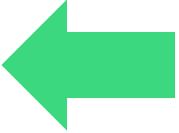
C) 5 or more

D) exactly 4

# Can you answer these questions?

2. In Regex, [a-n] stands for?

- A) Returns a match for any digit between 0 and 9
- B) Returns a match for any lower case character, alphabetically between a and n
- C) Returns a match for any two-digit numbers from 00 and 59
- D) Returns a match for any character EXCEPT a, r, and n



# Session Plan - 4

## 5.4 SIGNIFICANCE OF CHARACTER CLASSES & SPECIAL SEQUENCES

### 5.4.1 CHARACTER CLASSES

### 5.4.2 SPECIAL SEQUENCES

# Character Classes

The character classes are sets of characters or ranges of characters enclosed by square brackets [].

For example,

- **[0-9]** it means match any digit from 0 to 9.
- **[4-8]** it means match any digit from 4 to 8.
- **[A-Z]** it means match any letters from A to Z.

# Character Classes

Let's see some of the most common character classes and Its Behavior–

Character Class	Meaning/Role/Behaviour
[pqr]	[pqr]: Match the letter <b>p</b> or <b>q</b> or <b>r</b>
[pqr][st]	[pqr][st]: Match the letter p or q or r followed by either <b>s</b> or <b>t</b>
[^abc]	[^abc]: Match any letter except <b>a,b,or c</b>
[0-9]	[0-9]: Match any digit from <b>0</b> to <b>9</b>
[a-z]	[a-z]: Match any lowercase letters from <b>a</b> to <b>z</b>
[A-Z]	[A-Z]: Match any uppercase letters from <b>A</b> to <b>Z</b>
[a-zA-Z]	[a-zA-Z]: Match any lowercase or uppercase letter
[s-t1-9]	[s-t1-9]: Match the letter between <b>s</b> and <b>t</b> and digits from <b>1</b> to <b>9</b> but not t1
[a-zA-Z0-9_]	[a-zA-Z0-9_]: Match any alphanumeric character

# Character Classes - Example

**Example** – Using regular expression, check if email id is in correct format or not

**Solution** – Email - Id has been broken in three parts

**XYZ @ abes.ac.in**

**before @ part, after @part and after dot(.) part**

# Character Classes - Example

```
import re
pattern="@"
string1="a@a.com"
string2="a.com"
s1=re.findall(pattern,string1)
print(s1)
```

['@'] —————→ Output

```
import re
pattern="@\w\."
string1="a@a.com"
s1=re.findall(pattern,string1)
print(s1)
```

['@a.' ] —————→ Output

} Searching for @

} Searching for @ followed by  
any character or digit and  
dot(.)

```
import re
pattern="\w+@\w\.\w+"
string1="a@a.com"
s1=re.findall(pattern,string1)
print(s1)
```

['a@a.com']



Searching for any character  
before @, after @ and after  
dot(.)

# Character Classes - Example

Pattern to specify email-id

```
patternemail="[A-Za-z0-9\.\+\_]+@[A-Za-z0-9\.\_\-]+\.[a-zA-Z]*$"
test="anurag.mishra@abes.ac.in"           ← First email-id to check if valid or not
test1="anurag.mishra.com"                  ← Second email-id to check if valid or not
ab=re.match(patternemail,test)
ab1=re.match(patternemail,test1)
```

```
print(ab)
print(ab1)
```

```
<re.Match object; span=(0, 24), match='anurag.mishra@abes.ac.in'>
None
```

Second email-id not valid

First email-id valid

Output

# Can you answer these questions?

What will be the output of the given code?

- a) ['i', 'i', 'e', 'o']
- b) ['i', 'e', 'o']
- c) 4
- d) None

```
import re  
  
re.findall('[aeiou]', 'i like python')
```

# Can you answer these questions?

Complete the given program to check if a string has at least 1 zero.

- a) '0'
- b) '[0]' 
- c) '[0-9]'
- d) None

```
import re
pattern = '_____'
if re.search(pattern,'192 Patel nagar'):
    print("string has at least 1 zero")
else:
    print("string has No zero")
```

# Special Sequences

The **special sequence** represents the basic predefined character classes, which have a unique meaning.

They are written as a **backslash \** followed by **any character** and it has a special meaning.

For example,

- \d sequence is similar to character class [0-9] , which means match any digit from 0 to 9.
- \w sequence is similar to [a-zA-Z0-9\_], which means match any lowercase, uppercase, digit and underscore.

# Special Sequences

Some of the special sequences are given in the figure below

Special Sequences	Meaning/Role/Behaviour
\A	\A: Matches the specified characters at the beginning of the string
\Z	\Z: Matches the specified characters are at the end of the string
\d	\d: Matches the string contains digits
\D	\D: Matches the string does not contain digits
\s	\s: Matches the string contains a white space character
\S	\S: Matches the string doen not contains a white space character
\w	\w: Matches any characters from a to Z, digits from 0-9, and the " _ "
\W	\W: Matches the any characters not from [ a to Z, digits from 0-9, and the e " _ "]
\b	\b: Matches the specified characters are at the beginning or at the end
\B	\B: Matches the specified characters are present, but NOT at the start or at the end

# Special Sequence \A

The \A sequences only match the beginning of the string. It works the same as the carrot (^) metacharacter.

**Note:** If we do have a multi-line string, then \A will still match only at the beginning of the string, while the carrot (^) will match at the beginning of each new line of the string.

# Special Sequence \A

## Example –

```
import re

txt = "The ABESEC in Ghaziabad"

#Check if the string starts with "The":
```

```
x = re.findall("\AThe", txt)
y = re.findall("\A ABESEC", txt)
```

```
print(x)
print(y)
```

Pattern  
without Space

Pattern  
with Space

{'The'}  
[] Output

# Special Sequence \Z

Backslash Z (\Z) sequences only match the end of the string.

It works the same as the dollar (\$)  
meta-character.

```
import re

txt = "ABESEC in Ghaziabad"

#Check if the string ends with "Ghaziabad":

x = re.findall("Ghaziabad\Z", txt)
y = re.findall("ABESEC\Z", txt)
z = re.findall("Ghaziabad \Z", txt)

print(x)
print(y)
print(z)
```

Pattern  
without Space

Different Pattern  
With \Z

Pattern  
with Space

['Ghaziabad']  
[]  
[]

Output

# Special Sequence \d

**Backslash d or \d** matches any digits from 0 to 9 inside the target string.

This special sequence is to character class [0-9] .

```
import re

txt = "ABES 19th KM Stone, NH 24, Ghaziabad"

#Check if the string contains any digits (from 0-9):

x = re.findall("\d", txt)           Special  
Sequence

print(x)

['1', '9', '2', '4']             Output
```

# Special Sequence \D

**Backslash D or \D** is the exact opposite of \d. Any character in the target string that is not a digit would be the equivalent of the \D.

Also, we can write \D using **character class [^0-9]**.

```
import re

txt = "ABES NH 24"

#Return a match at every no-digit character:

x = re.findall("\D", txt)
```

Special  
Sequence

```
print(x)
```

['A', 'B', 'E', 'S', ' ', 'N', 'H', ' ']

Output

# Special Sequence \w

**Backslash w or \w** matches any alphanumeric character, also called a word character. This includes lowercase and uppercase letters, the digits 0 to 9, and the underscore character which is Equivalent to character **class [a-zA-z0-9\_]**.

```
import re

txt = "ABESEC NH_24"

#Return characters from a to Z, digits from 0-9 and _
x = re.findall("\w", txt)
y = re.findall("[a-zA-z0-9_]", txt)
print(x)
print(y)
```

Special  
Sequence

Character  
Class

['A', 'B', 'E', 'S', 'E', 'C', 'N', 'H', '\_', '2', '4']  
['A', 'B', 'E', 'S', 'E', 'C', 'N', 'H', '\_', '2', '4']

Output

# Special Sequence \W

**Backslash W or \W** is the exact opposite of \w, i.e., It matches any NON-alphanumeric character. \W same as character class **[^a-zA-Z0-9\_]**.

```
import re

txt = "ABESEC NH_24!"

#Returns the string which DOES NOT contain any word characters

x = re.findall("\W", txt)
y = re.findall("[^a-zA-Z0-9_]", txt)
print(x)
print(y)
```

Special  
Sequence

Character  
Class

[' ', '!']  
[' ', '!']

Output

# Special Sequence \s

**Backslash s or \s** matches any whitespace characters (Spaces, tab character (\t) ,newline character (\n) etc) inside the target string.

```
import re

txt = "ABESEC NH_24 "

#Return a match at every white-space character

x = re.findall("\s", txt)
y = re.findall("[ \t\n\x0b\r\f]", txt)
print(x)
print(y)
```

Special Sequence

Character Class

[ ' ', ' ' ]  
[ ' ', ' ' ]

Output

# Special Sequence \S

**Backslash S or \S** is the exact opposite of \s, and it matches any character which is not a whitespace character.

```
import re

txt = "ABESEC NH_24 "

#Returns the string DOES NOT contain a white space character

x = re.findall("\S", txt)
y = re.findall("[^ \t\n\x0b\f]", txt)

print(x)
print(y)
```

Special  
Sequence

Character  
Class

[ 'A', 'B', 'E', 'S', 'E', 'C', 'N', 'H', '\_', '2', '4' ]  
 [ 'A', 'B', 'E', 'S', 'E', 'C', 'N', 'H', '\_', '2', '4' ]

Output

# Special Sequence \b

**Backslash b or \b** matches the empty string, but only at the beginning or end of a word.

```
import re

txt = "ABESEC NH_24 or ABESEC Ghzb "

#Check if "ABESEC" is present at the beginning of a WORD
#Check if "EC" is present at the end of a WORD

x = re.findall(r"\bABESEC", txt)
y = re.findall(r"\b ABESEC", txt)
z = re.findall(r"EC\b", txt)

print(x)
print(y)
print(z)
```

Special Sequence  
for start

Space with Special  
Sequence

Special Sequence  
for end

[ 'ABESEC', 'ABESEC' ]  
[ ' ABESEC' ]  
[ 'EC', 'EC' ]

} Output

# Special Sequence \B

**Backslash B or \B** is the exact opposite of \b, it matches the empty string, but only when it is not at the beginning or end of a word.

```
import re

txt = "TheABESEC NH_24 is an Engineering College"

#Check if "ABESEC" is present, but NOT at the beginning of a word
#Check if "Engineer" is present, but NOT at the end of a word
#Check if "ring" is present, but NOT at the end of a word

x = re.findall(r"\BABESEC", txt)          Special Sequence  
for start
y = re.findall(r"Engineer\B", txt)
z = re.findall(r"ring\B", txt)           Special Sequence  
for end

print(x)
print(y)
print(z)

['ABESEC']
['Engineer']
[]
```

Output

# Example

Write a Python program to find all four characters long word in a string.

```
import re
text = 'ABES EC Campus:1, NH24, Ghaziabad, UP'
print(re.findall(r"\b\w{4}\b", text))
```

['ABES', 'NH24']



Output

# Example

**Write a Python program to find all three, four, five and six characters long words in a string.**

```
import re
text = 'ABES EC Campus:1, NH24, Ghaziabad, UP'
print(re.findall(r"\b\w{3,6}\b", text))
```

['ABES', 'Campus', 'NH24']

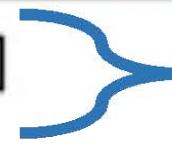
Output

# Example

Write a Python program to find all words which are at least 5 characters long in a string.

```
import re
text = 'ABES EC Campus:1, NH24, Ghaziabad, UP'
print(re.findall(r"\b\w{5,}\b", text))
```

[ 'Campus', 'Ghaziabad' ]



Output

# Session Plan - Day 5

**5.2 Group(),  
groups(),  
sub(),  
split(),  
compile().**

# More Regex methods

## Group () or Groups() –

We would use group(num)or groups() function of match object to get matched expression.

### Syntax -



# Cont..

**Example 1: Write a program to print the username, org\_name and domain from a emailID.**

'()' parenthesis are used to define a specific group

```

import re
test='pythongroup@abes.ac.in'
match_object = re.match(r'(\w+)@(\w+)\.(\w+)', test)

print(match_object.group())
print(match_object.group(0))
print(match_object.group(1))
print(match_object.group(2))
print(match_object.group(3))

print(match_object.groups())
print(match_object.group(1, 2, 3))

```

for entire match

for the first subgroup

for the second subgroup

for the third subgroup

for a tuple of all matched subgroups

Output

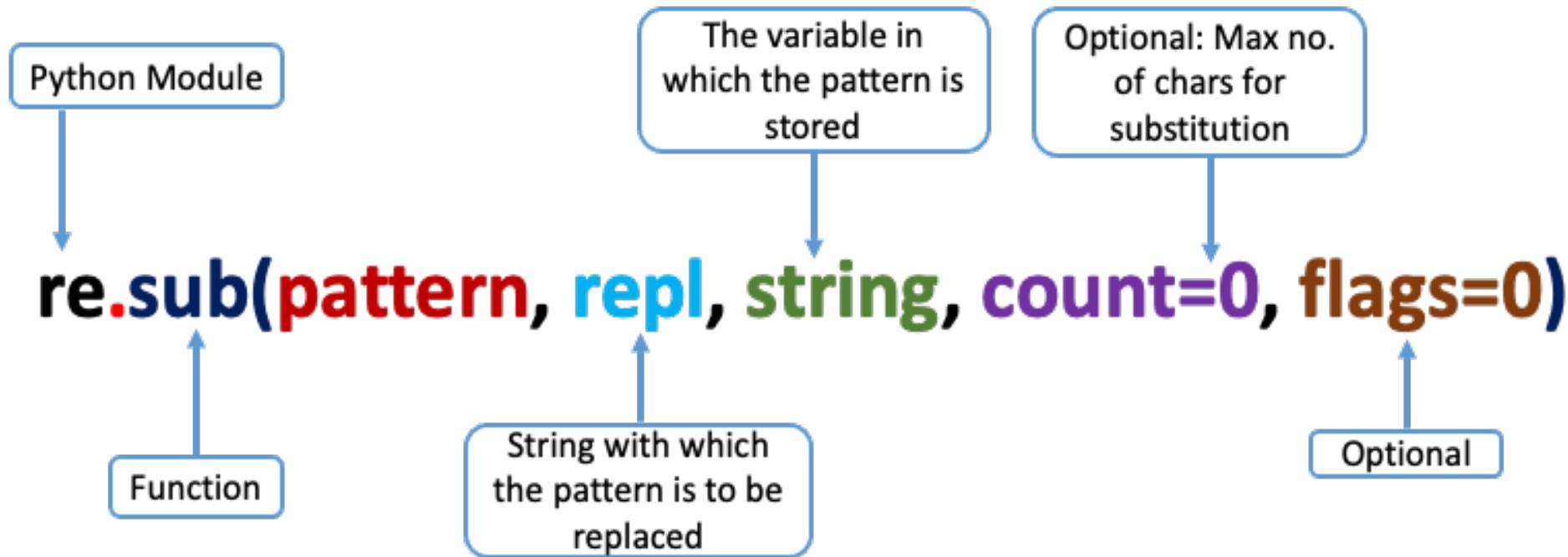
```

pythongroup@abes.ac
pythongroup@abes.ac
pythongroup
abes
ac
('pythongroup', 'abes', 'ac')
('pythongroup', 'abes', 'ac')

```

# re.sub() method

The `re.sub()` method replaces instances of a certain sub-string with another sub-string.



# re.sub() method

Example - Write a program to demonstrate sub() method in regular expression.

```
import re

text = "ABESEC Campus 1, NH 24, GZB, UP"
output1 = re.sub("\s", "-", text) ← Replace space by -
print(output1)

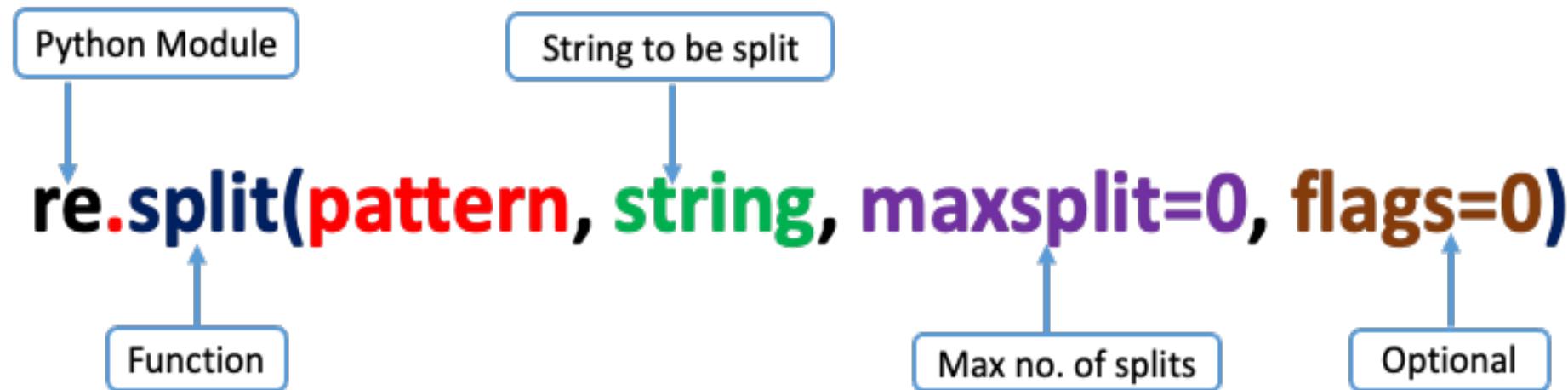
output2 = re.sub("\s", ":", text, 2) ← Replace the first 2
print(output2) occurrences of
space by :
```

ABESEC-Campus-1,-NH-24,-GZB,-UP  
ABESEC:Campus:1, NH 24, GZB, UP

Output

# re.split() method

The `re.split()` method split the string by the occurrences of the regex pattern, returning a list containing the resulting substrings.



# re.split() method

**Example 1:** Write a program to demonstrate split() method in regular expression.

```
import re

text = "ABESEC: NH24, GZB, UP"

x = re.split("\s", text)

y = re.split("\s", text, maxsplit=2) ←
print(x)
print(y)
```

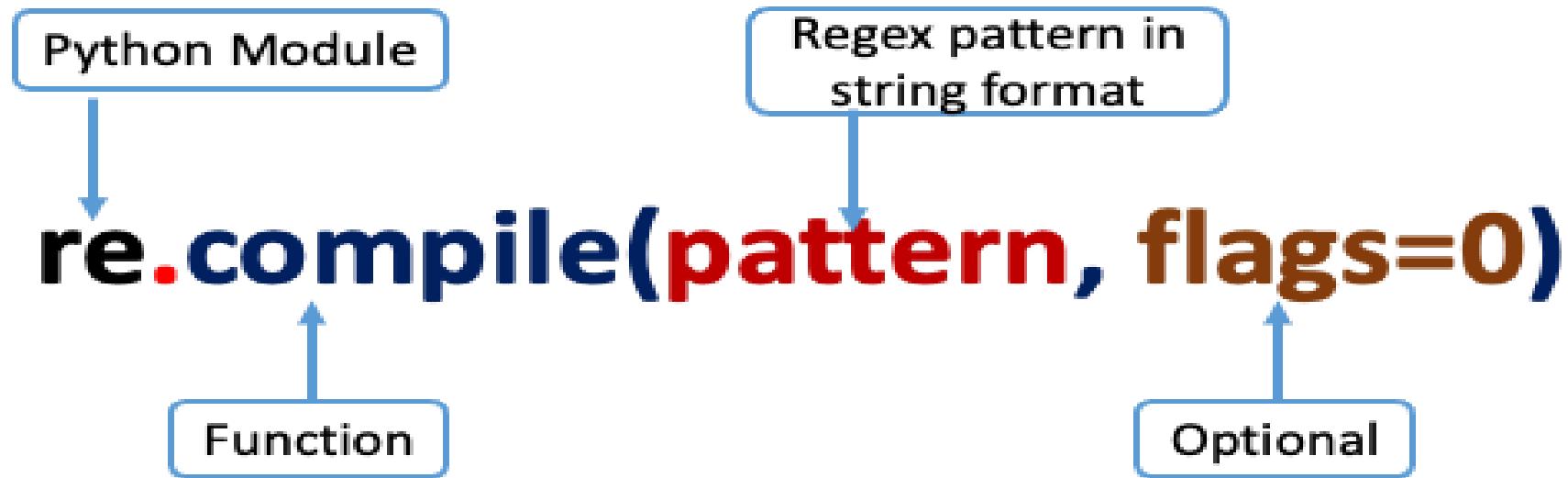
Most two splits

[ 'ABESEC:', 'NH24,', 'GZB,', 'UP' ]  
[ 'ABESEC:', 'NH24,', 'GZB, UP' ]

Output

# re.compile() method

The `re.compile()` method is to compile the regex pattern into pattern object (`re.Pattern`), which can be used for matching later.



# re.compile() method

**Example:** Write a program to demonstrate compile() method in regular expression.

```
import re

# Target String
text = "ABESEC:Campus1, NH 24, GZB"
pattern = r"\d"

pattern_object = re.compile(pattern)    re.pattern object
print(type(pattern_object))

result = pattern_object.findall(text) ← Use Pattern object returned
print(result)                          by the compile() method to
                                         match a regex pattern.
```

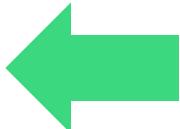
<class 're.Pattern'>  
['1', '2', '4']

} Output

# Can you answer these questions?

Complete the given program to check if a string has at least 1 zero.

- a) ['Python']
- b) []
- c) ['I Python']
- d) Error



```
import re
txt = 'You Like Python'
s = re.findall("(\\AI |Python\\Z)",txt)
print(s)
```

# Can you answer these questions?

Complete the given program to check if a string has at least 1 zero.

- a) ['Python']
- b) ['I'] 
- c) ['I Python']
- d) Error

```
import re
txt = 'I Love Python but I Like C'
s = re.findall("(\\AI|Python\\Z)",txt)
print(s)
```

# Can you answer these questions?

Which of the following creates a pattern object?

a) `re.create(str)`

b) `re.regex(str)`

c) `re.compile(str)`



d) `re.assemble(str)`

# Can you answer these questions?

What will be the output of the following Python code?

```
re.split(',', 'I Love Python, but I Like C')
```

- a) ['I Love Python', ' but I Like C'] 
- b) ['I', 'Love', 'Python', 'but', 'I', 'Like', 'C']
- c) ['I', 'Love', 'Python']
- d) Error

# Summary

# References

1. <https://docs.python.org/3/tutorial/controlflow.html>
2. Think Python: An Introduction to Software Design, Book by Allen B. Downey
3. Head First Python, 2nd Edition, by Paul Barry
4. Python Basics: A Practical Introduction to Python, by David Amos, Dan Bader, Joanna Jablonski, Fletcher Heisler
5. <https://fresh2refresh.com/python-tutorial/python-jump-statements/>
6. <https://tutorialsclass.com/python-jump-statements/>

# Thank You

---