Unit-01
Programming with Python

✅ Outline

- ✓ Introduction to python
- ✓ Installing python
- ✓ Hello World program using python
- ✓ Data types
- ✓ Variables
- ✓ Expressions
- ✓ Functions
- ✓ String
- ✓ List
- ✓ Tuple
- ✓ Set
- ✓ Dictionary
- ✓ Functions

Introduction to Python

▸ Python is an **open source, interpreted, high-level**, **general-purpose** programming language.

▸ Python's design philosophy emphasizes **code readability** with its notable use of significant **whitespace**.

▸ Python was conceived in the late **1980s** as a successor to the **ABC language**.

▸ Python was Created by **Guido van Rossum** and first released in **1991**.

▸ **Python 2.0**, released in **2000**,

  ↳ introduced features like list comprehensions and a garbage collection system with reference counting.

▸ **Python 3.0** released in **2008** and current version of python is **3.8.3** (as of June-2020).

  ↳ The Python 2 language was officially discontinued in 2020

Why Python?

▸ Python has many advantages

➥ Easy to learn

➥ Less code

➥ Syntax is easier to read

➥ Open source

➥ Huge amount of additional open-source libraries

Some libraries listed below.

▪ **matplotib** for plotting charts and graphs

▪ **BeautifulSoup** for HTML parsing and XML

▪ And many more..

Difference Between Python Modules, Packages, Libraries, and Frameworks

Python Modules: A module is basically a bunch of related code saved in a file with the extension .py

For example:

random module to generate pseudo-random number generators for various distributions.

datetime module to manipulate date and time data.

re module to detect and parse regular expressions in Python.

Python Packages: Python packages are basically a directory of a collection of modules. Packages allow the hierarchical structure of the module namespace.

For example:

NumPy is the fundamental Python package for scientific computing.

pandas is a Python package for fast and efficient processing of tabular data, time series, matrix data, etc.

pytest provides a variety of modules to test new code, including small unit tests or complex functional tests.

Python Libraries: A library is an umbrella term referring to a reusable chunk of code. Usually, a Python library contains a collection of related modules and packages. However, it is often assumed that while a package is a collection of modules, a library is a collection of packages.

For Example:

Matplotlib library is a standard library for generating data visualizations in Python. It supports building basic two-dimensional graphs as well as more complex animated and interactive visualizations.

PyTorch is an open-source deep-learning library built by Facebook's AI Research lab to implement advanced neural networks and cutting-edge research ideas in industry and academia.

pygame provides developers with tons of convenient features and tools to make game development a more intuitive task.

Beautiful Soup is a very popular Python library for getting data from the web. The modules and packages inside this library help extract useful information from HTML and XML files.

Python Frameworks: Similar to libraries, Python frameworks are a collection of modules and packages that help programmers to fast track the development process. However, frameworks are usually more complex than libraries. Also, while libraries contain packages that perform specific operations, frameworks contain the basic flow and architecture of the application.

For Example:

Django is a Python framework for building web applications with less coding. With all the necessary features included by default, developers can focus on their applications rather than dealing with routine processes.

Flask is a web development framework that is known for its lightweight and modular design. It has many out-of-the-box features and is easily adaptable to specific requirements.

Bottle is another lightweight framework for web development that was originally meant for building APIs. Its unique features are that it has no dependencies other than the Python Standard Library and it implements everything in a single source file.

Python Keywords

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

Installing Python

▶ For Windows & Mac:
  ↪ To install python in windows you need to download installable file from https://www.python.org/downloads/
  ↪ After downloading the installable file you need to execute the file.

▶ For Linux :
  ↪ For ubuntu 16.10 or newer
    ▪ sudo apt-get update
    ▪ sudo apt-get install python3.8

▶ To verify the installation
  ↪ Windows :
    ▪ python --version
  ↪ Linux :
    ▪ **python3** --version (linux might have python2 already installed, you can check python 2 using **python --version**)

▶ Alternatively we can use anaconda distribution for the python installation
  ↪ http://anaconda.com/downloads
  ↪ Anaconda comes with many useful inbuilt libraries.

Literal, Variable, & Identifier

▶ Literal: A literal is notation for representing a fixed (const) value

Example: a = 3.14 fixed value

▶ Variable: A variable is storage location associated with a symbolic name. abc = 5

▶ Identifier:The identifier is only used to identify an entity uniquely in a program at the time of execution whereas, a variable is a name given to a memory location, that is used to hold a value.

▶ Ex: student = "ICT"

S.No.	Identifier	Variable
1.	It is used to name a variable, a function, a class, a structure, a union.	It is used to give a name to a memory location that holds a value.
2.	Identifiers are created assign a name to an entity.	Variable is created to assign a unique name to a specific memory location.
3.	All identifiers are not variables.	All the variables names are identifiers.
4.	Identifier can take more number of characters.	Variable takes less number of characters.

Python multiple assignment

x = y = z = 99

print(x,y) or print(x) print(y) print(z)

Another example of multiple assignment

a,b,c = 5,6,7

print(a,b,c) or print(a) print(b) print(c)

Plus and concatenation operation on the variables

X = 10 y = 20

Print(x+y)

P = "Hello"

Q = "World"

Print(p+ " " + q)

Hello World using Python

▸ To write python programs, we can use any text editors or IDE (Integrated Development Environment), Initially we are going to use Visual Studio Code.

▸ Create new file in editor, save it as **first.py** (Extensions for python programs will be .py)

```
first.py
1  print("Hello World from python")
```

Python line does not end with ;

▸ To run the python file open command prompt and change directory to where your python file is

```
D:\>cd B.E
D:\B.E>cd 5th
D:\B.E\5th>cd "Phython 2020"
D:\B.E\5th\Phython 2020>cd Demo
```

▸ Next, run python command (python filename.py)

```
D:\B.E\5th\Phython 2020\Demo>python first.py
Hello World from python
```

Data types in Python

Name	Type	Description
		Data Types
Integer	int	Whole number such as 0,1,5, -5 etc..
Float	float	Numbers with decimal points such as 1.5, 7.9, -8.2 etc..
String	str	Sequence of character (Ordered) such as "ICT", 'Marwadi University', etc..
Boolean	bool	Logical values indicating **T**ure or **F**alse (T and F here are capital in python)
		Data Structures
List	list	**Ordered** Sequence of objects, will be represented with **square** brackets **[ ]** Example : [18, "ICT", True, 102.3]
Tuple	tup	**Ordered immutable** sequence of objects, will be represented with **round** brackets **( )** Example : (18, "ICT", True, 102.3)
Set	set	**Unordered** collection of **unique** objects, will be represented with the **curly** brackets **{ }** Example : { 18, "ICT", True, 102.3 }
Dictionary	dict	**Unordered key : value** pair of objects , will be represented with **curly** brackets **{ }** Example : { "college": "Marwadi University", "code": "054" }

Variables in Python

‣ A Python variable is a reserved memory location to store values.

‣ Unlike other programming languages, Python has no command for declaring a variable.

‣ A variable is created the moment you first assign a value to it.

‣ Python uses Dynamic Typing so,
  - We need not to specify the data types to the variable as it will internally assign the data type to the variable according to the value assigned.
  - we can also reassign the different data type to the same variable, variable data type will change to new data type automatically.
  - We can check the current data type of the variable with **type(variablename)** in-built function.

‣ Rules for variable name
  - Name can not start with digit
  - Space not allowed
  - Can not contain special character
  - Python keywords not allowed
  - **Should** be in lower case

Example of Python variable

▸ Example :

demo.py

```python
1   x = 10
2   print(x)
3   print(type(x))
4
5   y = 123.456
6   print(y)
7
8   x = "Marwadi University"
9   print(x)
10  print(type(x))
```

Reassign same variable to hold different data type

Run in terminal

```
1   python demo.py
```

Output

```
1   10
2   int
3   123.456
4   Marwadi University
5   str
```

String in python

▸ String is **Ordered Sequence of character** such as "Marwadi", 'college' etc..

▸ Strings are **arrays of bytes** representing **Unicode** characters.

▸ String can be represented as single, double or triple quotes.

▸ String with **triple** Quotes allows **multiple** lines.

▸ String in python is **immutable**.

▸ Square brackets can be used to access elements of the string, Ex. "Marwadi"[1] = a, characters can also be accessed with reverse index like "Marwadi"[-1] = i.

String index

```
        x = " M  a  r  w  d  i "
    index =    0  1  2  3  4  5
Reverse index =  -6 -5 -4 -3 -2 -1
```

Python - Slicing Strings

b = "Hello, World!"

print(b[2:5]) :-

print(b[:5]) :-

print(b[2:]) :-

print(b[::-1]) :-

print(b[-1::]) :-

print(b[::1]) :-

print(b[1::]) :-

print(b[::2]) :-

Print(b[-5:-2])

'a' in 'program' :- True

'at' not in 'battle' :- False

Python program to slice the given string using positive indexing

Python program to slice the given string using negative indexing

String Operators in Python

▶ Assignment operator: "=."

▶ Concatenate operator: "+."

▶ String repetition operator: "*."

▶ String slicing operator: "[]"

▶ String comparison operator: "==" & "!="

▶ Membership operator: "in" & "not in"

▶ Escape sequence operator: "\."

▶ String formatting operator: "%" & "{}"

Assignment Operator "="

```python
string1 = "hello"
```

Concatenate Operator "+"

```python
str1 = "hello"
str2 = "world "
str = str1+str2
```

String Repetition Operator "*"

```python
string1 = "helloworld "
print(string1*2)
```

String slicing operator "[]"

```python
string1 = "helloworld"
print(string1[1])
```

String Comparison Operator "==" & "!="

```python
string1 = "hello"
string2 = "hello, world"
print(string1==string4)
print(string1!=string4)
```

Membership Operator "in" & "not in"

```python
string1 = "helloworld"
print("w" in string1)
print("t" not in string1)
```

Escape Sequence Operator "\"

```python
string = "Hello world I am from "India""
print(string)

string = "Hello world I am from \"India\""
print(string)
```

String Formatting Operator "%"

```python
name = "india" age = 19 marks = 20.56
string1 = 'Hey %s' % (name)
print(string1)
string2 = 'my age is %d' % (age)
print(string2)
string3= 'Hey %s, my age is %d' % (name, age)
print(string3)
string3= 'Hey %s, my subject mark is %f' % (name, marks)
print(string3)
```

Task:

1.Compare Two Strings

2. Join Two or More Strings

3. Create a string made of the first, middle and last character

str1 = "James"

Jms

4. Create a string made of the middle three characters

str2 = "JaSonAy"

Son

String functions in python

▶ Python has lots of built-in methods that you can use on strings, we are going to cover some frequently used methods for string like

- **len()**
- count()
- capitalize(), lower(), upper()
- istitle(), islower(), isupper()
- find(), rfind(), replace()
- index(), rindex()
- Methods for validations like
  - isalpha(), isalnum(), isdecimal(), isdigit()
- strip(), lstrip(), rstrip()
- Etc..

▶ **Note** : len() is not the method of the string but can be used to get the length of the string

lendemo.py

```
1  x = "Darshan"
2  print(len(x))
```

Output : 7 (length of "Darshan")

String methods (cont.)

▶ **count**() method will returns the number of times a specified value occurs in a string.

countdemo.py

```
1  x = "Darshan"
2  ca = x.count('a')
3  print(ca)
```

Output : 2 (occurrence of 'a' in "Darshan")

▶ **title**(), **lower**(), **upper**() will returns capitalized, lower case and upper case string respectively.

changecase.py

```
1  x = "Marwadi University"
2  c = x.title()
3  l = x.lower()
4  u = x.upper()
5  print(c)
6  print(l)
7  print(u)
```

Output : Marwadi University

Output : marwadi university

Output : MARWADI UNIVERSITY

String methods (cont.)

▸ **istitle**(), **islower**(), **isupper**() will returns True if the given string is capitalized, lower case and upper case respectively.

checkcase.py
```
1  x = 'Marwadi University'
2  c = x.istitle()
3  l = x.islower()
4  u = x.isupper()
5  print(c)
6  print(l)
7  print(u)
```

Output : *False*

Output : *True*

Output : *False*

▸ **strip**() method will remove whitespaces from both side of the string and returns the string.

stripdemo.py
```
1  x = '     Darshan  '
2  f = x.strip()
3  print(f)
```

Output : *Darshan* (without space)

▸ **rstrip**() and **lstrip**() will remove whitespaces from right and left side respectively.

String methods (cont.)

▸ **find**() method will search the string and returns the index at which they find the specified value

finddemo.py
```
1  x = 'Marwadi University'
2  f = x.find('a')
3  print(f)
```

Output : 1 (occurrence of "a' in x)

▸ **rfind**() will search the string and returns the last index at which they find the specified value

rfinddemo.py
```
1  x = 'Marwadi University'
2  r = x.rfind('a')
3  print(r)
```

Output : 4 (last occurrence of 'a' in x)

▸ **Note** : **find**() and **rfind**() will **return -1** if they are unable to find the given string.

▸ **replace**() will replace str1 with str2 from our string and return the updated string

replacedemo.py
```
1  x = 'Marwadi University, Rajkot'
2  r = x.replace('Rajkot','RAJKOT')
3  print(r)
```

Output :
"Marwadi University, RAJKOT"

String methods (cont.)

▶ **index**() method will search the string and returns the index at which they find the specified value, but if they are unable to find the string it will raise an exception.

```
1  x = 'Marwadi University, rajkot, india'
2  f = x.index('in')
3  print(f)
```

Output : 8 (occurrence of 'Un' in x)

▶ **rindex**() will search the string and returns the last index at which they find the specified value , but if they are unable to find the string it will raise an exception.

```
1  x = 'Marwadi University, rajkot, india'
2  r = x.rindex('in')
3  print(r)
```

Output : 29 (last occurrence of 'in' in x)

▶ Note : **find**() and **index**() are almost same, the only difference is if **find**() is unable to find the string it will return -1 and if **index**() is unable to find the string it will raise an exception.

String methods (cont.)

▶ **isalnum**() method will return true if all the characters in the string are alphanumeric (i.e either alphabets or numeric).

isalnumdemo.py
```
1  x = 'Marwadi123'
2  f = x.isalnum()
3  print(f)
```

Output : *True*

▶ **isalpha**() and **isnumeric**() will return true if all the characters in the string are only alphabets and numeric respectively.

▶ **isdecimal**() will return true is all the characters in the string are decimal.

isdecimaldemo.py
```
1  x = '123'
2  r = x.isdecimal()
3  print(r)
```

Output : *True*

String print format

▸ **str.format()** is one of the *string formatting methods* in Python3, which allows multiple substitutions and value formatting.

▸ This method lets us concatenate elements within a string through positional formatting.

strformat.py

```
1  x = '{} University, rajkot'
2  y = x.format('Marwadi')
3  print(y)
4  print(x.format('ICT'))
```

Output : *Marwadi University, rajkot*

Inline function call
Output : *ICT University, rajkot*

▸ We can specify multiple parameters to the function

strformat.py

```
1  x = '{} University, {}'
2  y = x.format('Marwadi','Rajkot')
3  print(y)
4  print(x.format('ABCD','XYZ'))
```

Output : *Marwadi University,Rajkot*

Inline function call
Output : *ABCD University, XYZ*

String print format (cont.)

▸ We can specify the order of parameters in the string

strformat.py

```
1  x = '{1} University, {0}'
2  y = x.format('Marwadi','Rajkot')
3  print(y)
4  print(x.format('ABCD','XYZ'))
```

Output : *Rajkot University,Marwadi*

Inline function call
Output : *XYZ University,ABCD*

▸ We can also specify alias within the string to specify the order

strformat.py

```
1  x = '{UniversityName} institute, {CityName}'
2  print(x.format(UniversityName='Marwadi',CityName='Rajkot'))
```

Output : *Marwadi University,Rajkot*

▸ We can format the decimal values using format method

strformat.py

```
1  per = (438 / 500) * 100
2  x = 'result = {r:3.2f} %'.format(r=per)
3  print(x)
```

Output : *result = 87.60 %*

width   precision

Task

▶ Append new string in the middle of a given string

s1 = "Ault"

s2 = "Kelly"

Expected Output:

AuKellylt

Data structures in python

▸ There are four built-in data structures in Python - *list, dictionary, tuple and set.*

Name	Type	Description
List	list	**Ordered** Sequence of objects, will be represented with **square** brackets **[ ]** Example : [18, "College", True, 102.3]
Dictionary	dict	**Unordered key : value** pair of objects , will be represented with **curly** brackets **{ }** Example : { "college": "Marwadi", "code": "054" }
Tuple	tup	**Ordered immutable** sequence of objects, will be represented with **round** brackets **( )** Example : (18, "Marwadi", True, 102.3)
Set	set	**Unordered** collection of **unique** objects, will be represented with the **curly** brackets **{ }** Example : { 18, "Marwadi", True, 102.3 }

▸ Lets explore all the data structures in detail…

List

▸ List is a mutable ordered sequence of objects, duplicate values are allowed inside list.

▸ List will be represented by square brackets [ ].

▸ Python does not have array, List can be used similar to Array.

```
list.py
1  my_list = ['Marwadi', 'institute', 'rajkot']
2  print(my_list[1])
3  print(len(my_list))
4  my_list[2] = "rajkot"
5  print(my_list)
6  print(my_list[-1])
```

Output : *institute* (List index starts with 0)

Output : *3* (length of the List)

Output : *['Marwadi', 'institute', 'rajkot']*
Note : *spelling of rajkot is updated*

Output : *rajkot* (-1 represent last element)

▸ We can use slicing similar to string in order to get the sub list from the list.

```
list.py
1  my_list = ['Marwadi', 'institute', 'rajkot','gujarat','INDIA']
2  print(my_list[1:3])
```

Output : *['institute', 'rajkot']*
Note : *end index not included*

List methods

▸ append() method will add element at the end of the list.

appendlistdemo.py

```
1  my_list = ['Marwadi', 'institute', 'rajkot']
2  my_list.append('gujarat')
3  print(my_list)
```

Output : *['Marwadi', 'institute', 'rajkot', 'gujarat']*

▸ insert() method will add element at the specified index in the list

insertlistdemo.py

```
1  my_list = ['Marwadi', 'institute', 'rajkot']
2  my_list.insert(2,'of')
3  my_list.insert(3,'engineering')
4  print(my_list)
```

Output : *['Marwadi', 'institute', 'of', 'engineering', 'rajkot']*

▸ extend() method will add one data structure (List or any) to current List

extendlistdemo.py

```
1  my_list1 = ['Marwadi', 'institute']
2  my_list2 = ['rajkot','gujarat']
3  my_list1.extend(my_list2)
4  print(my_list1)
```

Output : *['Marwadi', 'institute', 'rajkot', 'gujarat']*

List methods (cont.)

▶ pop() method will remove the last element from the list and return it.

poplistdemo.py

```
1  my_list = ['Marwadi', 'institute','rajkot']
2  temp = my_list.pop()
3  print(temp)
4  print(my_list)
```

Output : *rajkot*

Output : *['Marwadi', 'institute']*

▶ remove() method will remove first occurrence of specified element

removelistdemo.py

```
1  my_list = ['Marwadi', 'institute', 'Marwadi','rajkot']
2  my_list.remove('Marwadi')
3  print(my_list)
```

Output : *['institute', 'Marwadi', 'rajkot']*

▶ clear() method will remove all the elements from the List

clearlistdemo.py

```
1  my_list = ['Marwadi', 'institute', 'Marwadi','rajkot']
2  my_list.clear()
3  print(my_list)
```

Output : *[]*

▶ index() method will return first index of the specified element.

List methods (cont.)

▸ count() method will return the number of occurrence of the specified element.

countlistdemo.py

```
1  my_list = ['Marwadi', 'institute', 'Marwadi'...]
2  c = my_list.count('Marwaadi')
3  print(c)
```

Output : *2*

▸ reverse() method will reverse the elements of the List

reverselistdemo.py

```
1  my_list = ['Marwadi', 'institute','rajkot']
2  my_list.reverse()
3  print(my_list)
```

Output : *['rajkot', 'institute','Marwadi']*

▸ sort() method will sort the elements in the List

sortlistdemo.py

```
1  my_list = ['Marwadi', 'college','of','enginnering','rajkot']
2  my_list.sort()
3  print(my_list)
4  my_list.sort(reverse=True)
5  print(my_list)
```

Output : *['college', 'Marwadi', 'enginnering', 'of', 'rajkot']*

Output : *['rajkot', 'of', 'enginnering', 'Marwadi', 'college']*

1. Maximum and Minimum K elements in Tuple Using slicing + sorted()

The original tuple is : (5, 20, 3, 7, 6, 8)
The extracted values : (3, 5, 6, 7, 8, 20)

2. Find repeated items in a tuple.

Input:(2, 4, 5, 6, 2, 3, 4, 4, 7)
Output: 3

Tuple

▸ Tuple is a immutable ordered sequence of objects, duplicate values are allowed inside list.

▸ Tuple will be represented by round brackets ( ).

▸ Tuple is similar to List but List is mutable whearas Tuple is immutable.

tupledemo.py

```
1  my_tuple = ('Marwadi','institute','of','engine
2  print(my_tuple)
3  print(my_tuple.index('engineering'))
4  print(my_tuple.count('of'))
5  print(my_tuple[-1])
```

Output : *('Marwadi', 'institute', 'of', 'engineering', 'of', 'rajkot')*

Output : *3* (index of 'engineering')

Output : *2*

Output : *rajkot*

Dictionary

▸ Dictionary is a unordered collection of key value pairs.

▸ Dictionary will be represented by curly brackets { }.

▸ Dictionary is mutable.

syntax

```
my_dict = { 'key1':'value1', 'key2':'value2' }
```

Key value is seperated by **:**

Key value pairs is seperated by **,**

dictdemo.py

```
1  my_dict = {'college':"Marwadi", 'city':"rajkot",'type':"engineering"}
2  print(my_dict['college'])
3  print(my_dict.get('city'))
```

values can be accessed using key inside square brackets
as well as using get() method
Output : *Marwadi*

rajkot

Dictionary methods

▶ keys() method will return list of all the keys associated with the Dictionary.

keydemo.py

```
1  my_dict = {'college':"Marwadi", 'city':"rajkot",'type':"engineering"}
2  print(my_dict.keys())
```

Output : *['college', 'city', 'type']*

▶ values() method will return list of all the values associated with the Dictionary.

valuedemo.py

```
1  my_dict = {'college':"Marwadi", 'city':"rajkot",'type':"engineering"}
2  print(my_dict.values())
```

Output : *['Marwadi', 'rajkot', 'engineering']*

▶ items() method will return list of tuples for each key value pair associated with the Dictionary.

itemsdemo.py

```
1  my_dict = {'college':"Marwadi", 'city':"rajkot",'type':"engineering"}
2  print(my_dict.items())
```

Output : *[('college', 'Marwadi'), ('city', 'rajkot'), ('type', 'engineering')]*

Set

▸ Set is a unordered collection of unique objects.

▸ Set will be represented by curly brackets { }.

tupledemo.py
```
1  my_set = {1,1,1,2,2,5,3,9}
2  print(my_set)
```

Output : *{1, 2, 3, 5, 9}*

▸ Set has many in-built methods such as add(), clear(), copy(), pop(), remove() etc.. which are similar to methods we have previously seen.

▸ Only difference between Set and List is that Set will have only unique elements and List can have duplicate elements.

Task

▶ Write a Python program to convert a tuple to a string

▶ Swap two tuples in Python

```
tuple1 = (11, 22) tuple2 = (99, 88)
```
tuple1: (99, 88) tuple2: (11, 22)

▶ **Access value 20 from the tuple**

```
tuple1 = ("Orange", [10, 20, 30], (5, 15, 25))
```
Modify the tuple

```
tuple1 = (11, [22, 33], 44, 55)
```
tuple1: (11, [222, 33], 44, 55)

Operators in python

▸ We can segregate python operators in the following groups
  ↳ Arithmetic operators
  ↳ Assignment operators
  ↳ Comparison operators
  ↳ Logical operators
  ↳ Identity operators
  ↳ Membership operators
  ↳ Bitwise operators

▸ We will discuss some of the operators from the given list in detail in some of next slides.

Arithmetic Operators

▶ Note : consider A = 10 and B = 3

Operator	Description	Example	Output
+	Addition	A + B	13
-	Subtraction	A - B	7
/	Division	A / B	3.3333333333333335
*	Multiplication	A * B	30
%	Modulus return the remainder	A % B	1
//	Floor division returns the quotient	A // B	3
**	Exponentiation	A ** B	10 * 10 * 10 = 1000

Logical Operators

▸ Note : consider A = 10 and B = 3

Operator	Description	Example	Output
and	Returns True if both statements are true	A > 5 **and** B < 5	True
or	Returns True if one of the statements is true	A > 5 **or** B > 5	True
not	Negate the result, returns True if the result is False	**not** (A > 5)	False

Identity & Member Operators

▸ Identity Operator

▸ Note : consider A = [1,2], B = [1,2] and C=A

Operator	Description	Example	Output
is	Returns True if both variables are the same object	A is B A is C	FALSE TRUE
is not	Returns True if both variables are **different** object	A is not B	TRUE

▸ Member Operator

▸ Note : consider A = 2 and B = [1,2,3]

Operator	Description	Example	Output
in	Returns True if a sequence with the specified value is present in the object	A in B	TRUE
not in	Returns True if a sequence with the specified value is not present in the object	A not in B	FALSE

If statement

▶ if statement is written using the **if** keyword followed by **condition** and **colon**(**:**) .

▶ Code to execute when the condition is true will be ideally written in the next line with **Indentation** (white space).

▶ Python relies on indentation to define scope in the code (Other programming languages often use curly-brackets for this purpose).

Syntax
```
1  if some_condition :
2      # Code to execute when condition is true
```

if statement ends with **:**

Indentation (tab/whitespace) at the beginning

ifdemo.py
```
1  x = 10
2
3  if x > 5 :
4      print("X is greater than 5")
```

Run in terminal
```
1  python ifdemo.py
```

Output
```
1  X is greater than 5
```

If else statement

```
1  if some_condition :
2      # Code to execute when condition is true
3  else :
4      # Code to execute when condition is false
```

ifelsedemo.py

```
1  x = 3
2
3  if x > 5 :
4      print("X is greater than 5")
5  else :
6      print("X is less than 5")
```

Run in terminal

```
1  python ifelsedemo.py
```

Output

```
1  X is less than 5
```

If, elif and else statement

```
1  if some_condition_1 :
2      # Code to execute when condition 1 is true
3  elif some_condition_2 :
4      # Code to execute when condition 2 is true
5  else :
6      # Code to execute when both conditions are false
```

```
1  x = 10
2
3  if x > 12 :
4      print("X is greater than 12")
5  elif x > 5 :
6      print("X is greater than 5")
7  else :
8      print("X is less than 5")
```
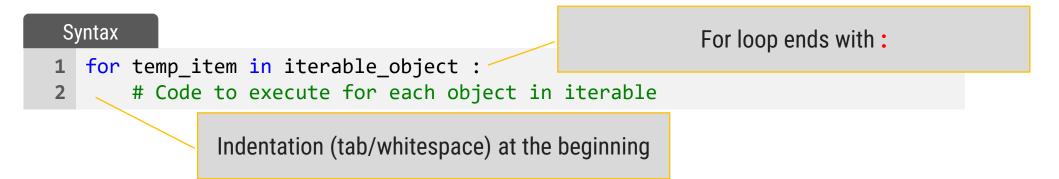
```
1  python ifelifdemo.py
```

```
1  X is greater than 5
```

For loop in python

▶ Many objects in python are **iterable,** meaning we can iterate over every element in the object.
  ↳ such as every elements from the List, every characters from the string etc..

▶ We can use for loop to execute block of code for each element of iterable object.

Syntax

```
1  for temp_item in iterable_object :
2      # Code to execute for each object in iterable
```

For loop ends with **:**

Indentation (tab/whitespace) at the beginning

fordemo1.py

```
1  my_list = [1, 2, 3, 4]
2  for list_item in my_list :
3      print(list_item)
```

Output :
```
1
2
3
4
```

fordemo2.py

```
1  my_list = [1,2,3,4,5,6,7,8,9]
2  for list_item in my_list :
3      if list_item % 2 == 0 :
4          print(list_item)
```

Output :
```
2
4
6
8
```

For loop (tuple unpacking)

▸ Sometimes we have nested data structure like List of tuples, and if we want to iterate with such list we can use tuple unpacking.
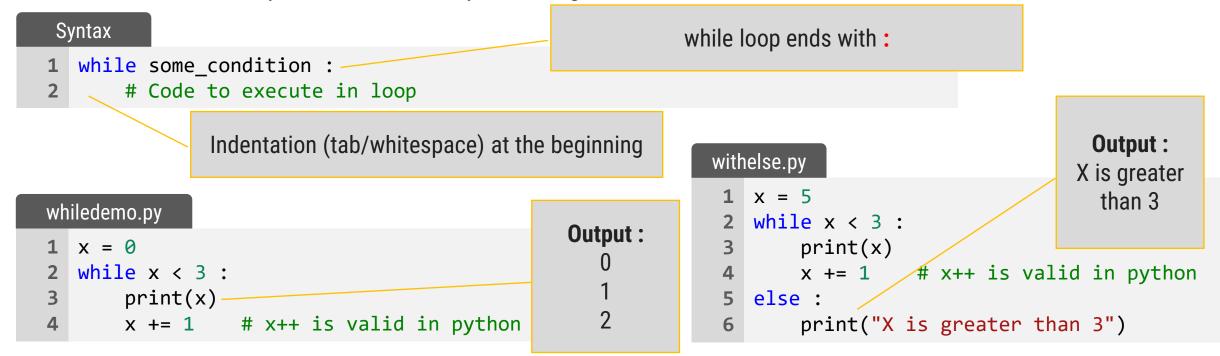
withouttupleunapacking.py
```
1  my_list = [(1,2,3), (4,5,6), (7,8,9)]
2  for list_item in my_list :
3      print(list_item[1])
```

Output :
2
5
8

withtupleunpacking.py
```
1  my_list = [(1,2,3), (4,5,6), (7,8,9)]
2  for a,b,c in my_list :
3      print(b)
```

This technique is known as tuple unpacking

Output :
2
5
8

▸ range() function will create a list from 0 till (not including) the value specified as argument.

rangedemo.py
```
1  my_list = range(5)
2  for list_item in my_list :
3      print(list_item)
```

Output :
0
1
2
3
4

While loop

▸ While loop will continue to execute block of code until some condition remains True.

▸ For example,
  ➥ while felling hungry, keep eating
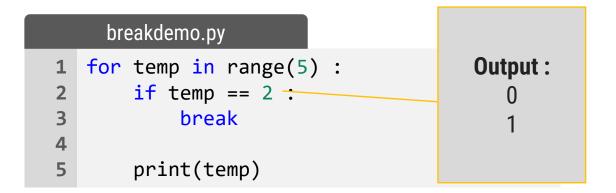 ➥ while have internet pack available, keep watching videos

Syntax

```
1  while some_condition :
2      # Code to execute in loop
```

while loop ends with **:**

Indentation (tab/whitespace) at the beginning

whiledemo.py

```
1  x = 0
2  while x < 3 :
3      print(x)
4      x += 1    # x++ is valid in python
```

Output :
0
1
2

withelse.py

```
1  x = 5
2  while x < 3 :
3      print(x)
4      x += 1      # x++ is valid in python
5  else :
6      print("X is greater than 3")
```

Output :
X is greater than 3

break, continue & pass keywords

▶ break : Breaks out of the current closest enclosing loop.

breakdemo.py

```
1  for temp in range(5) :
2      if temp == 2 :
3          break
4
5      print(temp)
```

Output :
0
1

▶ continue : Goes to the top of the current closest enclosing loop.

continuedemo.py

```
1  for temp in range(5) :
2      if temp == 2 :
3          continue
4
5      print(temp)
```

Output :
0
1
3
4

▶ Pass : Does nothing at all, will be used as a placeholder in conditions where you don't want to write anything.

passdemo.py
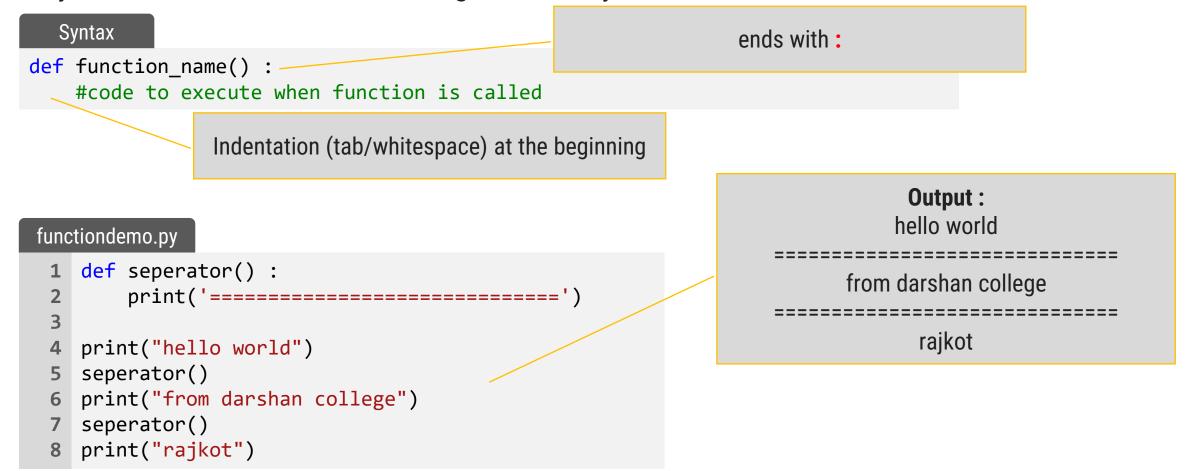
```
1  for temp in range(5) :
2      pass
```

Output : (nothing)

Functions in python

▸ Creating clean repeatable code is a key part of becoming an effective programmer.

▸ A function is a block of code which only runs when it is called.

▸ In Python a function is defined using the def keyword:

Syntax

```
def function_name() :
    #code to execute when function is called
```

ends with **:**

Indentation (tab/whitespace) at the beginning

functiondemo.py

```
1  def seperator() :
2      print('==============================')
3
4  print("hello world")
5  seperator()
6  print("from darshan college")
7  seperator()
8  print("rajkot")
```

Output :
hello world
==============================
from darshan college
==============================
rajkot

Function (cont.) (DOCSTRIGN & return)

▶ Doc string helps us to define the documentation about the function within the function itself.

```
def function_name() :
    '''
        DOCSTRING: explains the function
        INPUT: explains input
        OUTPUT: explains output
    '''

    #code to execute when function is called
```

Enclosed within triple quotes

▶ **return statement** : return allows us to assign the output of the function to a new variable, return is use to send back the result of the function, instead of just printing it out.

whiledemo.py

```
1  def add_number(n1,n2) :
2      return n1 + n2
3
4  sum1 = add_number(5,3)
5  sum2 = add_number(6,1)
6  print(sum1)
7  print(sum2)
```

Output :
8
7