Unit-01

Getting started with Python

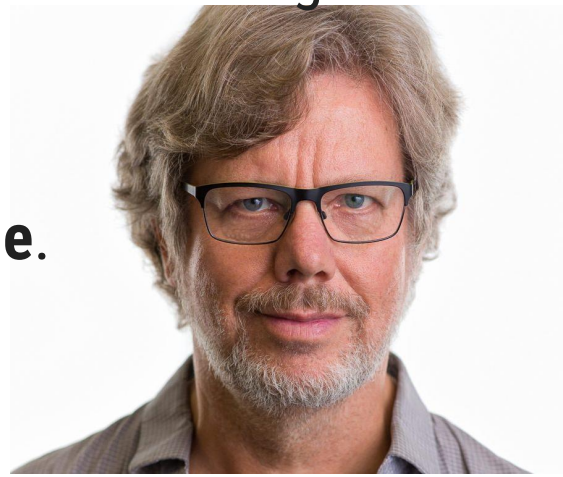
• *Prof. Keyur Prajapati*

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 is **dynamically typed** and **garbage-collected** language**.**
- 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
    - **NumPy** for scientific computing
    - **pandas** for performing data analysis
    - **SciPy** for engineering applications, science, and mathematics
    - **Scikit** for machine learning
    - **Django** for server-side web development
    - And many more..

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.

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 “MBIT”, ‘college’, “Anand” 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, “MBIT”, True, 102.3]
Tuple	tup	**Ordered immutable** sequence of objects, will be represented with **round** brackets **( )** Example : (18, “MBIT”, True, 102.3)
Set	set	**Unordered** collection of **unique** objects, will be represented with the **curly** brackets **{ }** Example : { 18, “MBIT”, True, 102.3 }
Dictionary	dict	**Unordered key : value** pair of objects , will be represented with **curly** brackets **{ }** Example : { “college”: “MBIT”, “code”: “063” }

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

```
1  x = 10
2  print(x)
3  print(type(x))
4
5  y = 123.456
6  print(y)
7
8  x = “Madhuben & Bhanubhai Patel Institute of Technology"
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  Madhuben & Bhanubhai Patel Institute of Technology
5  str
```

String in python

- String is **Ordered Sequence of character** such as “Cricket”, ‘college’, “આણંદ ” 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. “Cricket”[1] = r, characters can also be accessed with reverse index like “Cricket”[-1] = t.

String index

```
          x = "  C  r  i  c  k  e  t  "
      index =    0  1  2  3  4  5  6
Reverse index =    0 -6 -5 -4 -3 -2 -1
```

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 = “Cricket"
2 print(len(x))
```

Output : 7 (length of “Cricket”)

String methods (cont.)

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

countdemo.py

```
1  x = “Cricket"
2  ca = x.count(‘c')
3  print(ca)
```

Output : 2 (occurrence of ‘c’ in “Cricket”)

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

changecase.py

```
1  x = “MBIT college, Anand"
2  c = x.title()
3  l = x.lower()
4  u = x.upper()
5  print(c)
6  print(l)
7  print(u)
```

Output : MBIT college, Anand

Output : mbit college, anand

Output : MBIT COLLEGE, ANAND

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 = ‘mbit college, anand'
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 = '     MBIT   '
2 f = x.strip()
3 print(f)
```

Output : *MBIT* (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 = ‘mbit institute, anand, india'
2 f = x.find(‘in')
3 print(f)
```

Output : 6 (occurrence of ‘in’ in x)

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

rfinddemo.py

```
1 x =  ‘mbit institute, anand, india '
2 r = x.rfind('in')
3 print(r)
```

Output : 24 (last occurrence of ‘in’ 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 =  ‘mbit institute, anand, india'
2 r = x.replace('india','INDIA')
3 print(r)
```

Output :
“mbit institute, anand, INDIA”

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.

indexdemo.py

```
1 x = ‘mbit institute, anand, india'
2 f = x.index('in')
3 print(f)
```

Output : 5 (occurrence of ‘in’ 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.

rindexdemo.py

```
1 x = ‘mbit institute, anand, india'
2 r = x.rindex('in')
3 print(r)
```

Output : 23 (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 = ‘mbit123'
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 = '123.5'
2 r = x.isdecimal()
3 print(r)
```

Output : *True*

- **Note** : **isnumeric**() and **isdigit**() are almost same, you suppose to find the difference as Home work assignment for the string methods.

String Slicing

- We can get the substring in python using string slicing, we can specify start index, end index and steps (colon separated) to slice the string.

syntax

```
x = ’madhuben and bhanubhai patel institute of technology, anand, gujarat, INDIA’
subx = x[startindex:endindex:steps]
```

endindex will not be included in the substring

strslicedemo.py

```
1  x =  ’madhuben and bhanubhai patel institute of technology, anand, gujarat, INDIA '
2  subx1 = x[0:7]
3  subx2 = x[49:55]
4  subx3 = x[66:]
5  subx4 = x[::2]
6  subx5 = x[::-1]
7  print(subx1)
8  print(subx2)
9  print(subx3)
10 print(subx4)
11 print(subx5)
```

Output: madhube

Output : ogy, a

Output : at, INDIA

Output : mdue n hnba ae nttt ftcnlg,aad uaa,IDA

Output : AIDNI ,tarajug ,dnana ,ygolonhcet fo etutitsni letap iahbunahb dna nebuhdam

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 = '{} institute, anand'
2 y = x.format('mbit')
3 print(y)
4 print(x.format('ABCD'))
```

Output : *mbit institute, anand*

Inline function call
Output : *ABCD institute, anand*

- We can specify multiple parameters to the function

strformat.py

```
1 x = '{} institute, {}'
2 y = x.format('mbit','anand')
3 print(y)
4 print(x.format('ABCD','XYZ'))
```

Output : *mbit institute, anand*

Inline function call
Output : *ABCD institute, XYZ*

String print format (cont.)

- We can specify the order of parameters in the string

strformat.py

```
1 x = '{1} institute, {0}'
2 y = x.format('mbit','anand')
3 print(y)
4 print(x.format('ABCD','XYZ'))
```

Output : *anand institute, mbit*

Inline function call
Output : *XYZ institute, ABCD*

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

strformat.py

```
1 x = '{collegename} institute, {cityname}'
2 print(x.format(collegename='mbit',cityname='anand'))
```

Output : *mbit institute, anand*

- 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

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, “mbit”, True, 102.3]
Dictionary	dict	**Unordered key : value** pair of objects , will be represented with **curly** brackets **{ }** Example : { “college”: “mbit”, “code”: “063” }
Tuple	tup	**Ordered immutable** sequence of objects, will be represented with **round** brackets **( )** Example : (18, “mbit”, True, 102.3)
Set	set	**Unordered** collection of **unique** objects, will be represented with the **curly** brackets **{ }** Example : { 18, “mbit”, 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 = [‘mbit', 'institute', ‘and']
2 print(my_list[1])
3 print(len(my_list))
4 my_list[2] = “anand"
5 print(my_list)
6 print(my_list[-1])
```

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

Output : *3* (length of the List)

Output : *[‘mbit', 'institute', ‘anand']*
Note : *spelling of anand is updated*

Output : *anand* (-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 = [‘mbit', 'institute', ‘anand','gujarat','INDIA']
2 print(my_list[1:3])
```

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

List methods

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

appendlistdemo.py

```
1 my_list = [‘mbit', 'institute', ‘anand']
2 my_list.append('gujarat')
3 print(my_list)
```

Output : *[‘mbit', 'institute', ‘anand', 'gujarat']*

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

insertlistdemo.py

```
1 my_list = [‘mbit', 'institute', ‘anand']
2 my_list.insert(2,'of')
3 my_list.insert(3,'engineering')
4 print(my_list)
```

Output : *[‘mbit', 'institute', 'of', 'engineering', ‘anand']*

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

extendlistdemo.py

```
1 my_list1 = [‘mbit', 'institute']
2 my_list2 = [‘anand','gujarat']
3 my_list1.extend(my_list2)
4 print(my_list1)
```

Output : *[‘mbit', 'institute', ‘anand', ‘gujarat']*

List methods (cont.)

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

poplistdemo.py

```
1 my_list = [‘mbit', 'institute',‘anand']
2 temp = my_list.pop()
3 print(temp)
4 print(my_list)
```

Output : *anand*

Output : *[‘mbit', 'institute‘]*

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

removelistdemo.py

```
1 my_list = [‘mbit', 'institute', ‘mbit',‘anand']
2 my_list.remove(‘mbit')
3 print(my_list)
```

Output : *['institute', ‘mbit', 'anand']*

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

clearlistdemo.py

```
1 my_list = [‘mbit', 'institute', ‘mbit',‘anand']
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 = [‘mbit', 'institute', ‘mbit','rajkot']
2 c = my_list.count(‘mbit')
3 print(c)
```

Output : *2*

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

reverselistdemo.py

```
1 my_list = [‘mbit', 'institute',‘anand']
2 my_list.reverse()
3 print(my_list)
```

Output : *[‘anand', ‘institute',‘mbit']*

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

sortlistdemo.py

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

Output : *['anand', ‘college', 'enginnering', ‘mbit', ‘of']*

Output : *['of', ‘mbit', 'enginnering', ‘college', ‘anand']*

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 whereas Tuple is immutable.

tupledemo.py

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

Output : *(‘mbit', 'institute', 'of', 'engineering', 'of', ‘anand')*

Output : *3* (index of ‘engineering’)

Output : *2*

Output : *anand*

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':“mbit", 'city':“anand",'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 : *mbit*

anand

Dictionary methods

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

keydemo.py

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

Output : *[‘mbit', ‘anand', 'engineering']*

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

itemsdemo.py

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

Output : *[('college', ‘mbit'), ('city', ‘anand'), ('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.

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

Syntax

```
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

Syntax

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

ifelifdemo.py

```
x = 10

if x > 12 :
    print("X is greater than 12")
elif x > 5 :
    print("X is greater than 5")
else :
    print("X is less than 5")
```

Run in terminal

```
python ifelifdemo.py
```

Output

```
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 feeling 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 mbit college")
7  seperator()
8  print(“anand")
```

Output :

```
hello world
============================
from mbit college
============================
anand
```

Function (cont.) (DOCSTRING & return)

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

Syntax

```
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