

Day-01: Introduction to Python

Topics to be covered

1. Introduction to Python
2. Varieties of Python
3. Installation of Python
4. Setting Environment Variable
5. Basic Data Types
6. First Python Program

Introduction to Python

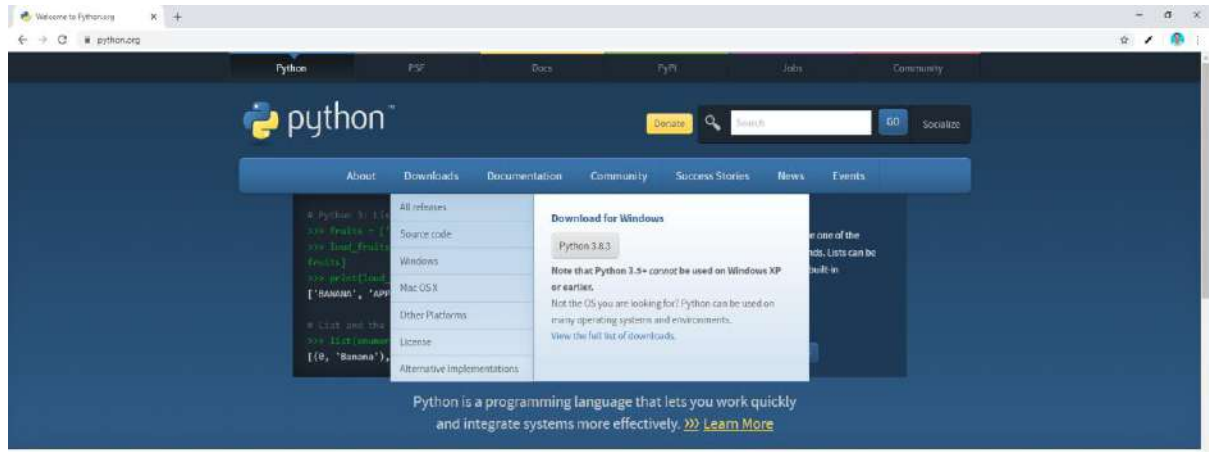
- It was created by Guido van Rossum in Netherland, and released in 1991.
- The name "Python" was adopted from the Rossum's favourite comedy series "Monty Python's Flying Circus".
- It is a high level general purpose programming language.
- It is compiled to byte code and executed in Python Virtual Machine.
- It is suitable for use as a scripting language, Web application implementation language, etc.
- It has a strong structuring constructs (nested code blocks, functions, classes, modules, and packages) and the use of objects and object oriented programming, enables us to write clear, logical applications for small and large tasks.
- Python is interpreted language.
- Python is available as Free and Open Source.
- Python run on different platform like Windows , Linux , Unix etc

Varieties of Python

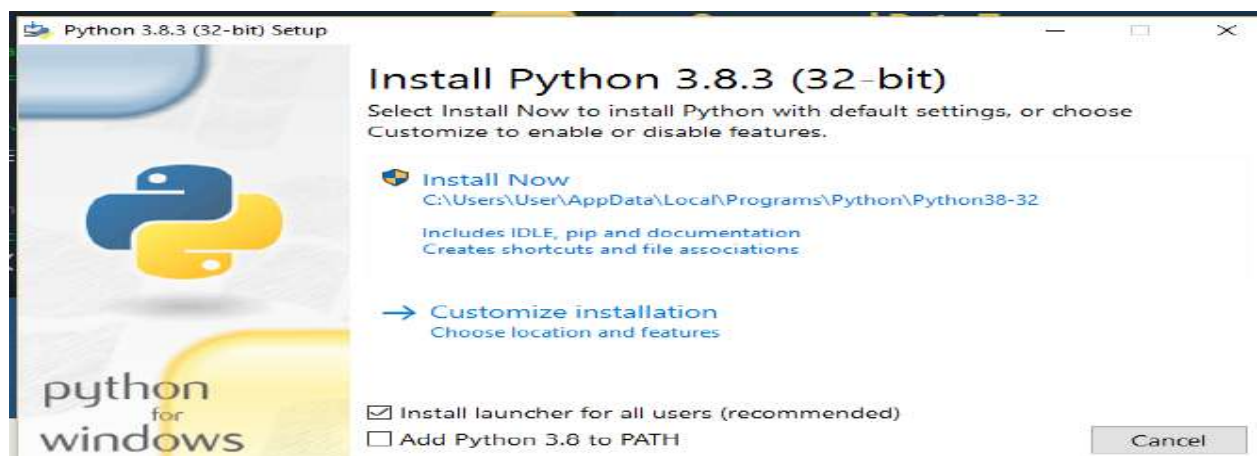
- CPython Standard Python 2.x implemented in C.
- Jython Python for the Java environment <http://www.jython.org/>
- PyPy Python with a JIT compiler and stackless mode <http://pypy.org/>
- Stackless Python with enhanced thread support and microthreads etc. <http://www.stackless.com/>
- IronPython Python for .NET and the CLR <http://ironpython.net/>
- Python 3 – The new, new Python. This is intended as a replacement for Python 2.x. <http://www.python.org/doc/>.

Installation of Python on Windows OS

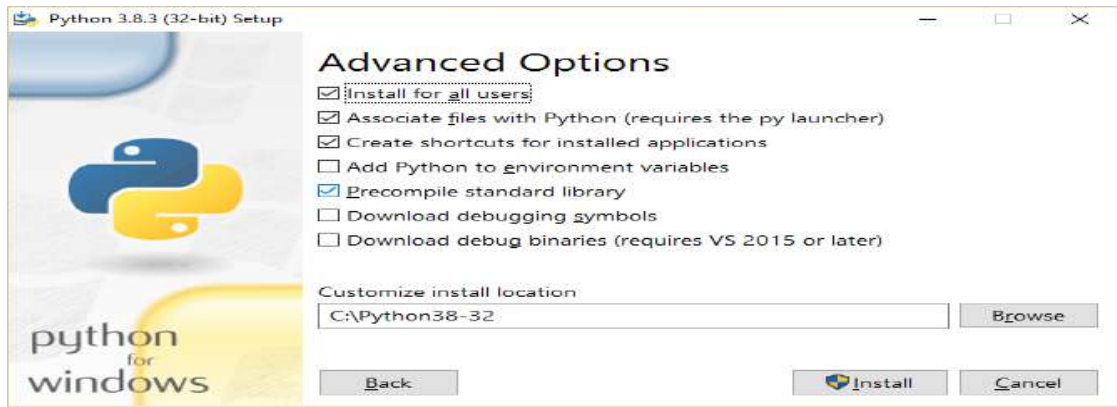
- Download the up-to-date source code, binaries, documentation available on the official website of Python <https://www.python.org/>.



- Python installer downloaded on to your system - Python-3.8.3.exe
- Click this icon and a installer screen shown as below



- Default option will also work, it automatically install the software at the location, it is showing.
- We can select the customize installation option, to install the software at the user specified location. - **C:\Python38-32**



Setting Environment Variable

- To add the Python directory to the path for a particular session in Windows –
 - At the command prompt, type
 - path %path%;C:\Python38-32 and press Enter.
 - Where C:\Python38-32 is the path of the Python directory.
- You can also select the – “**Add Python to Environment Variable**”, while installing the Python.
- After setting the path variable

C:\>python

Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (Intel)] on win32

Type "help", "copyright", "credits" or "license" for more information.

>>>

Python Programming

- **Comments** – Everything after # on a line is ignored.
- **Blocks and Indentation** – It follows the block structure and nested block structure with indentation.

ABC Block-1

ABC Block-2

ABC Block-3

- **Lines** – Statement separator is a semi colon, but needed only when there are more than one statement on a line.
- **File Extension**- Program have the File Extension “.py”.

- **Creating Variables**

- Variables are containers for storing data values.
- Python has no command for declaring a variable.
- A variable is created at the time, first value assign to it.
 - `x= 5` # Numeric Variable
 - `y= "Ajay"` # Character Variable
 - `print(x)`
 - `print(y)`

- **Rules for Variable Naming**

- A variable can have a short name (like x and y) or a more descriptive name (age, EmpName, total_volume).
- Rules for Python variables:
 - A variable name must start with a letter or the underscore character
 - A variable name cannot start with a number
 - A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
 - Variable names are case-sensitive
 - Example - age, Age and AGE are three different variable.

- **Reserve Words** – It can not be used as constant or any other identifier name.

and	exec	not	assert	finally	or	break
for	pass	class	from	print	continue	
global	raise	def	if	return	del	import
try	elif	in	while	else	is	with
except	yield	lambda				

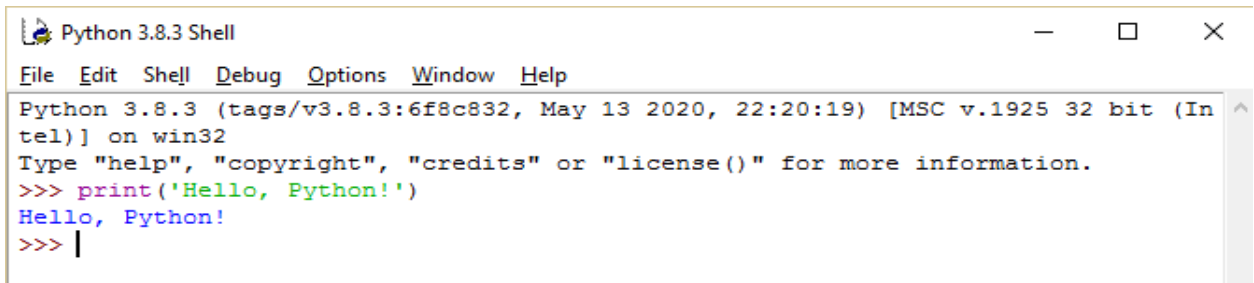
- **Basic Data Types**

- The data stored in memory can be of many types.
- For example, a person's age is stored as a numeric value and his or her address is stored as alphanumeric characters.

- Python has five standard data types –
 - Numbers
 - String
 - List
 - Tuple
 - Dictionary
- **Python Identifiers**
 - It is a name used to identify a variable, function, class, module or other objects in Python program.
 - An identifier starts with a letter (A to Z) or (a to z) or an underscore (_) followed by zero or more letters, underscores and digits (0 to 9).
 - Python does not allow the characters - @, \$, and % .
 - Valid Identifier- myvar, my_var , _my_var, myVar, MYVAR, myvar2
- **First Python Code**
 - Open the IDLE Python .
 - Type

```
print('Hello, Python!')
```
 - Press the Enter Key.



Python IDLE (Integrated Development and Learning Environment)

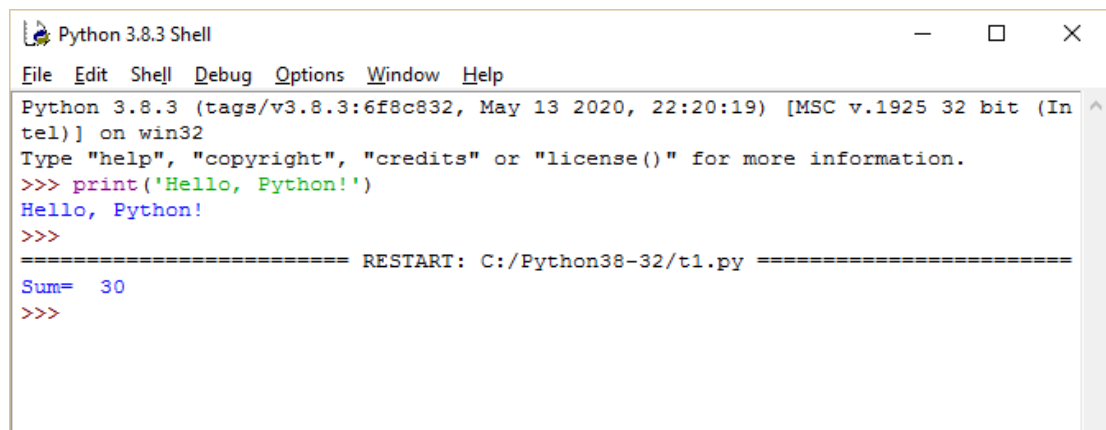
```
Python 3.8.3 Shell
File Edit Shell Debug Options Window Help
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> print('Hello, Python!')
Hello, Python!
>>> |
```

- **First Python Program**

- Open the IDLE Python as above.
- Open the File-> New File
- Type the program.

```
#program to print the sum of 2 number
n1=10
n2=20
n3=n1+n2
print('Sum= ', n3)
```

- Save the file with “.py” extension.
- Click the Run-> Module.



```
Python 3.8.3 Shell
File Edit Shell Debug Options Window Help
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> print('Hello, Python!')
Hello, Python!
>>>
===== RESTART: C:/Python38-32/t1.py =====
Sum= 30
>>>
```

Self-assessment for Day-01

1. Who has created the Python?
- Guido Van Rossum
 - Guido Vein Rasmi
 - Mark Elliot Zuckerberg
 - Jack Patrick Dorsey

Ans: a

2. Which is not the feature of Python?
- It is interpreter language.
 - It supports block and indentation for statements.
 - It does not support the OOPs Programming.
 - It is a general purpose programming language.

Ans: c

3. _____ is used to define the comments in the program.
- #
 - "
 - //
 - \$

Ans: a

4. _____ is not a reserve word.
- import
 - for
 - while
 - unknown

Ans: d

5. Which is invalid Identifiers.
- MYVAR
 - MY%VAR
 - MY_VAR
 - _MY_VAR

Ans. b

Choices must be unique and must not be like, "Both A & B", "All of the above", "None of the Above", etc.

Day-1

Q.1 Multiple Choice Questions

1. Who has created the Python?
 - a. Guido Van Rossum
 - b. Guido Vein Rasmi
 - c. Mark Elliot Zuckerberg
 - d. Jack Patrick Dorsey
2. Which is not the feature of Python?
 - a. It is interpreter language.
 - b. It supports block and indentation for statements.
 - c. It does not support the OOPs Programming.
 - d. It is a general purpose programming language.
3. _____ is used to define the comments in the program.
 - a. #
 - b. ‘
 - c. //
 - d. \$
4. _____ is not a reserve word.
 - a. import
 - b. for
 - c. while
 - d. unknown
5. Which is invalid Identifiers.
 - a. MYVAR
 - b. MY%VAR
 - c. MY_VAR
 - d. _MY_VAR

2. What is the output of following?

a) <pre>x, y, z = "Orange", "Banana", "Cherry" print(x) print(y) print(z)</pre>	b) <pre>x = y = z = "Orange" print(x) print(y) print(z)</pre>
c) <pre>x=10 y="Orange" print(x+y)</pre>	d) <pre>x=10 y="2020" print(x+y)</pre>

3. Write a program to print your name and age.
- 4.. Write a program to print the sum of three numbers?
Output - Sum of <10> , <20> , and <30> is <60>
<> replace is with your variable.
5. Write a program to print the area of rectangle.
Area = length * width

6. What is the output of following?

a) m1=30 m2=40 m3=50 print("Marks in Subject 1 :", m1) print("Marks in Subject 1 :", m2) print("Marks in Subject 1 :", m3)	b) a='Mango' print("You like the fruit - :", a) a=20 print("Result - :", a)
c) fruit='Mango' print('You like the fruit - : ', fruit)	d) fruit='Mango' print(' I like the ', fruit , ' , do you like it')

Day-02: Literals and Constants

Topics to be covered

1. Literals
2. Constants
3. Type of Literals
4. Python Program to demonstrate the Literals
5. Python Data Types
6. Python Program to demonstrate the Literals

Literals

- Literal is a raw data assigned to a variable or constant.
- Literals are immutable.
- Types of Literal
 - Text : str
 - Numeric : int, float, complex
 - Boolean Type : bool
 - Special : None
- Numeric Literal – 10 ,10.76 , 10+12j
- Text Literal – 'Mango' , "Mango" , ""Mango""
- Boolean Literal- True , False
-

Constants

- A constant is a type of variable whose value cannot be changed, during the execution of program.
- **In python, it does not prevent reassignment.**
- Constants are usually declared and assigned in a module.
- Module is a file containing variables, functions, etc which is imported to the main file.
- Usually, constants are written in all capital letters and underscores separating the words.

Types of Literals

- Numeric Literal
 - Numeric : int, float, complex
- Integer Literal
 - No fractional part allowed.
- Types of Integer Literal
 - Decimal Literal (base 10) - 10
 - Binary Literal (base 2) - **0b**1010
 - Octal Literal (base 8) - **0o**10
 - Hexadecimal Literal (base 16) - **0x**12c

Example

```
a=10
b=0b10
c=0o10
d=0x10
print('Decimal ', a)
print('Binary ', b)
print('Octal ', c)
print('Hexadecimal ', d)
```

Output

```
Decimal 10
Binary 2
Octal 8
Hexadecimal 16
```

What is the output, if a=0x1C ?

- **Float Literals**
 - Fractional part allowed.
- Types of Float Literal
 - Fixed Literal - 10.45
 - Scientific Literal – 10e2 , 10E2
 - 10e2 is equivalent to 10×10^2
 - 10e-2 is equivalent to 10×10^{-2}

Float Literal

10	E	2
Mantissa		Exponent
10	X	$10^2 = 1000$

- **Complex Literal**
 - It has real and imaginary part
 - 10+12j

Complex Literal

10	+	12j
Real		Imaginary

Example -1 For **Integer Literals**

```
a=0b10
b=0x10
c=0o10
d=a+b+c
print('Sum ',d)
```

Output

Sum 26

What is the output, if a=0x1c+0b1c ?

Example -2 For Float Literals

```
a=10.45
b=10E2
c=10E-2
print('Fixed ', a)
print('Scientific ',b)
print('Scientific ',c)
```

Output

```
Fixed  10.45
Scientific  1000.0
Scientific  0.1
```

What is the output, if a=0.1E3 ?

Example -2 For Complex Literal

```
d=10 +12j
print('Complex ', d)
print('Real Part ', d.real)
print('Imaginary Part ',d.imag)
```

Output

```
Complex (10+12j)
Real Part  10.0
Imaginary Part  12.0
```

String Literal

- It is a sequence of characters surrounded by quotes.
- Quotes can be single, double, or triple quotes for a string.
- A string that is prefixed with an **r** or **R** before the opening quotes is a “raw” string .
- Backslash (\) allow the line to be continued as single line.

- Example
 - `s1="This is Python"`
 - `s2= "P"`
 - `s3 = """This is a multiline string
with more than one line code."""`
 - `s4=r"First \n Second"`
 - `s5 ="This is \n
Python"`

Character pairs start with backslash(\)

`\n` = Newline

`\r` = Carriage Return

`\t` = Tab

`\\` = `\`

`\u` = Marks the start of a Unicode character code

`\"` or `\'` = A double or single quote character **without** ending the current string.

Example

```
s1="This is Python"
```

```
s2= "P"
```

```
s3 = """This is a multiline string  
with more than one line code."""
```

```
s4="First \n Second"
```

```
s5=r"First \n Second"
```

```
s6="This is \n  
Python"
```

```
print("s1 ",s1)
```

```
print("s2 ",s2)
```

```
print("s3 ",s3)
```

```
print("s4 ",s4)
```

```
print("s5 ",s5)
```

```
print("s6 ",s6)
```

Output

```
s1 This is Python
s2 P
s3 This is a multiline string
    with more than one line code.
s4 First
    Second
s5 First \n Second
s6 This is Python
```

- **Boolean literals**

- It can have two values: True or False.
- True has numerical value 1.
- False has numerical value 0.
- Example
 - `x1 = (1 == True)`
 - `x2 = (1 == False)`
 - `x3 = True + 4`
 - `x4 = False + 10`

Example

```
x1 = (1 == True)
x2 = (1 == False)
x3 = True + 4
x4 = False + 10
print("x1 is", x1)
print("x2 is", x2)
print("x3:", x3)
print("x4:", x4)
```

Output

```
x1 is True
x2 is False
x3: 5
x4: 10
```

Special literals

- Python has one special literal i.e. **None**.
- It to specify that the variable has not been created.
- Example
 - `x = 10`
 - `y = None`
- Memory allocated to variable when the literal assigned to the variable for the first time.

Example

```
x = 10
y = None
print("x is", x)
print("y is", y)
```

Output

```
x is 10
y is None
```

Python Data Types

- Data type allows the type of data stored in memory.
- For example,
 - A person's age is stored as a numeric value and
 - Address is stored as alphanumeric characters.
- Data Types defines the operations possible on them and the storage method for each of them.
- Variables can store data of different types, and different types can do different things.
- Python has five standard data types –
 - Numbers
 - String
 - List
 - Tuple
 - Dictionary

- Python has the following data types built-in under the categories:
 - Text Type : str
 - Numeric Types : int, float, complex
 - Sequence Types : list, tuple, range
 - Mapping Type : dict
 - Set Types : set
 - Boolean Type : bool

Example	Data Type
x = "Hello World"	Str
x = 20	Int
x = 20.5	Float
x = 1j	Complex
x = ["apple", "banana", "cherry"]	list
x = ("apple", "banana", "cherry")	Tuple
x = range(6)	Range
x = {"name" : "Ajay", "age" : 36}	Dict
x = {"apple", "banana", "cherry"}	Set
x = True	bool

Example

```
x1 = "Hello World"
x2 = 20
x3 = 20.5
x4 = 1j
x5 = True
print("String : ", x1)
print("Integer : ", x2)
print("Float : ", x3)
print("Complex : ", x4)
print("Bool : ", x5)
```

Output

String : Hello World

Integer : 20

Float : 20.5

Complex : 1j

Example-2

```
x6 = ["apple", "banana", "cherry"]
```

```
x7 = ("apple", "banana", "cherry")
```

```
x8 = range(6)
```

```
x9 = {"name": "Ajay", "age": 36}
```

```
x10 = {"apple", "banana", "cherry"}
```

```
print("List : ", x6)
```

```
print("Tuple : ", x7)
```

```
print("Range : ", x8)
```

```
print("Dictionary : ", x9)
```

```
print("Set : ", x10)
```

Output

List : ['apple', 'banana', 'cherry']

Tuple : ('apple', 'banana', 'cherry')

Range : range(0, 6)

Dictionary : {'name': 'Ajay', 'age': 36}

Set : {'banana', 'apple', 'cherry'}

Type Conversion

- Casting allows to specify a type on to a variable.

SNo.	Function	Description
1	int(x)	Converts x to an integer. base specifies the base if x is a string.
2	long(x)	Converts x to a long integer. base specifies the base if x is a string.
3	float(x)	Converts x to a floating-point number
4	complex(real [,imag])	Creates a complex number.
5	str(x)	Converts object x to a string representation.
6	tuple(s).	Converts s to a tuple
7	list(s)	Converts s to a list.
8	set(s)	Converts s to a set.
9	dict(d)	Creates a dictionary. d must be a sequence of (key,value) tuples.
10	chr(x)	Converts an integer to a character.

Integers:

<code>x = int(1)</code>	<code># x will be 1</code>
<code>y = int(2.8)</code>	<code># y will be 2</code>
<code>z = int("3")</code>	<code># z will be 3</code>

Floats:

<code>x = float(1)</code>	<code># x will be 1.0</code>
<code>y = float(2.8)</code>	<code># y will be 2.8</code>
<code>z = float("3")</code>	<code># z will be 3.0</code>
<code>w = float("4.2")</code>	<code># w will be 4.2</code>

Strings:

<code>x = str("s1")</code>	<code># x will be 's1'</code>
<code>y = str(2)</code>	<code># y will be '2'</code>
<code>z = str(3.0)</code>	<code># z will be '3.0'</code>

Day-02

Q.1 MCQ

- Which is an invalid Literal?
 - Test
 - Number
 - Boolean
 - Special
- Which is an invalid String Literal?
 - "This is Python"
 - "P"
 - 'Sum'
 - #Sum#
- Output of `print(0b1010)` is _____.
 - 1010
 - 0b1010
 - 10
 - b1010
- Which is an invalid data type?
 - List
 - Tuple
 - Array
 - String
- Which is invalid Constant Name?
 - MYVAR
 - MY%VAR
 - MY_VAR
 - _MY_VAR

6. What is the output of following?

a) <pre>y = 2.8 b = int(y) print(b)</pre>	b) <pre>x = "10" y=int(x) + 10 print(y)</pre>
c) <pre>x=10.73 y=int(x) print(x+y)</pre>	d) <pre>x=10E2 y=10+int(x) print(y)</pre>

7. Write a program to calculate the simple interest, if the principal, time and interest is given.
 $\text{SimpleInterest} = (\text{Principal} * \text{ROI} * \text{Time}) / 100$

8. What is the output of following?

a) <pre>m1=text1='hello\ user' print(m1)</pre>	b) <pre>msg=""welcome to NIELIT"" print(msg)</pre>
c) <pre>m1=text1='Apple \ Mango' print('You like the fruit - : ', m1)</pre>	d) <pre>fruit='Mango' print(fruit + 100)</pre>

9 Python program to exchange the values of two numbers without using a temporary variable.

10. Python program to calculate gross salary where gross salary=Basic+HRA+DA

In this HRA is 16% of Basic, DA is 12% of Basic

11. Write a program to calculate and print the sum of 3 non-integer numbers.

12. Write a program to calculate and print average of 3 numbers.

13. What is the output of following?

a) <code>y = 0x2c</code> <code>b = y + 10</code> <code>print(b)</code>	b) <code>x = 0o27</code> <code>y=x + 10</code> <code>print(y / 2)</code>
c) <code>x=0x2A + 0xAA</code> <code>y= 0o22</code> <code>print(x+y)</code>	d) <code>x= 0x2a + 0o2A</code> <code>y= x /2</code> <code>print(y)</code>

Day-03: Python Data Type and Operators

Topics to be covered

1. Python Data Type- String , List , Tuple, Dictionary
2. Python Operators
3. Arithmetic operators
4. Assignment operators
5. Comparison operators
6. Logical operators

Python Data Type - String

- Strings are identified as a contiguous set of characters represented in the quotation marks.
- Subsets of strings can be taken using the slice operator ([] and [:]) with indexes starting at 0 in the beginning of the string and working their way from -1 at the end.
- The plus (+) sign is the string concatenation operator.
- The asterisk (*) is the repetition operator.

Example

```
str = 'Hello NIELIT!'
print (str )          # Prints complete string
print (str[0])         # Prints first character of the string
print (str[2:5])       # Prints characters starting from 3rd to 5th
print (str[2:5:2])     # Prints characters starting from 3rd to 5th
print (str[2:])        # Prints string starting from 3rd character
print (str * 2)        # Prints string two times
print (str + "GKP")    # Prints concatenated string
```

Output

```
Hello NIELIT!
H
llo
lo
llo NIELIT!
Hello NIELIT!Hello NIELIT!
Hello NIELIT!GKP
```

Python Data Type - List

- A list contains items separated by commas and enclosed within square brackets ([]).
- Difference between C array and List is that all the items belonging to a list can be of different data type.
- The values stored in a list can be accessed using the slice operator ([] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end -1.
- The plus (+) sign is the list concatenation operator, and
- The asterisk (*) is the repetition operator.

Example

```
x = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
y = [123, 'john']
print(x)      # Prints complete list
print (x[0])  # Prints first element of the list
print (x[1:3]) # Prints elements starting from 2nd till 3rd
print (x[2:])  # Prints elements starting from 3rd element
print (y * 2)  # Prints list two times
print (x + y)  # Prints concatenated lists
```

Output

```
['abcd', 786, 2.23, 'john', 70.2]
abcd
[786, 2.23]
[2.23, 'john', 70.2]
[123, 'john', 123, 'john']
['abcd', 786, 2.23, 'john', 70.2, 123, 'john']
```


Python Data Type - Tuple

- A tuple is similar to the list.
- A tuple consists of a number of values separated by commas. Tuples are enclosed within parentheses.
- The plus (+) sign is the list concatenation operator, and
- The asterisk (*) is the repetition operator.
- The main differences between lists and tuples are:
- Lists are enclosed in brackets ([]) and their elements and size can be changed.
- While tuples are enclosed in parentheses (()) and cannot be updated.
- Tuples is an read-only lists.

Example

```
x = ( 'abcd', 786 , 2.23, 'john', 70.2 )
y = (123, 'john')
print(x)      # Prints complete tuple
print (x[0])  # Prints first element of the tuple
print (x[1:3]) # Prints elements starting from 2nd till 3rd
print (x[2:])  # Prints elements starting from 3rd element
print (y * 2)  # Prints list two times
print (x + y)  # Prints concatenated tuple
```

Output

```
('abcd', 786, 2.23, 'john', 70.2)
abcd
(786, 2.23)
(2.23, 'john', 70.2)
(123, 'john', 123, 'john')
('abcd', 786, 2.23, 'john', 70.2, 123, 'john')
```

Python Data Type - Dictionary

- Python's dictionaries is a key : value pair.
- A dictionary key are numbers or strings.
- Values can be any arbitrary Python object.
- Dictionaries are enclosed by curly braces { } and
- Values can be assigned and accessed using square braces [].

Example

```
d = {'name': 'john', 'code': 6734, 'dept': 'sales', 23: 45}
print(d)           # Prints complete dictionary
print(d.keys())    # Prints all the keys
print(d.values())  # Prints all the values
print(d["name"])
print(d[23])
```

Output

```
{'name': 'john', 'code': 6734, 'dept': 'sales', 23: 45}
dict_keys(['name', 'code', 'dept', 23])
dict_values(['john', 6734, 'sales', 45])
john
45
```

Python Operators

- Operators are used to perform operations on variables.
- Python divides the operators in the following groups:
 - 1. Arithmetic operators
 - 2. Assignment operators
 - 3. Comparison operators
 - 4. Logical operators
 - 5. Identity operators
 - 6. Membership operators
 - 7. Bitwise operators

1. Arithmetic Operator

- Arithmetic operators are used with numeric values to perform common mathematical operations.

Operator	Name	Example
+	Addition	$x + y$
-	Subtraction	$x - y$
*	Multiplication	$x * y$
/	Division	x / y
%	Modulus	$x \% y$
**	Exponentiation	$x ** y$
//	Floor division	$x // y$

Example-1

```
a = 21
b = 10
c = a + b
print("Line 1 - Value of c is ", c)
c = a - b
print("Line 2 - Value of c is ", c)
c = a * b
print("Line 3 - Value of c is ", c)
c = a / b
print("Line 4 - Value of c is ", c)
```

Output

```
Line 1 - Value of c is 31
Line 2 - Value of c is 11
Line 3 - Value of c is 210
Line 4 - Value of c is 2.1
```

Example -2

```
a = 21
b = 10
c = a % b
print("Line 5 - Value of c is ", c)
a = 2
b = 3
c = a**b
print("Line 6 - Value of c is ", c)
a = 10
b = 5
c = a//b
print("Line 7 - Value of c is ", c)
```

Output

Line 5 - Value of c is 1
Line 6 - Value of c is 8
Line 7 - Value of c is 2

2. Assignment Operator

- Assignment operators are used to assign values to variables.

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
=	x = 3	x = x 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

Example-1

```
a = 21
b = 10
c = 0
c = a + b
print ("Line 1 - Value of c is ", c)
c += a
print ("Line 2 - Value of c is ", c)
c = 10
c *= a
print ("Line 3 - Value of c is ", c)
c /= a
print ("Line 4 - Value of c is ", c)
c = 2
c %= a
print ("Line 5 - Value of c is ", c)
```

Output

Line 1 - Value of c is 31
Line 2 - Value of c is 52
Line 3 - Value of c is 210
Line 4 - Value of c is 10.0
Line 5 - Value of c is 2

Example -2

```
c=10
a=2
c **= a
print ("Line 6 - Value of c is ", c)
c /= a
print ("Line 7 - Value of c is ", c)
c=20
c >>= 2
print ("Line 8 - Value of c is ", c)
c=20
c <<= 1
print ("Line 9 - Value of c is ", c)
```

Output

Line 6 - Value of c is 100
Line 7 - Value of c is 50
Line 8 - Value of c is 5
Line 9 - Value of c is 40

3. Comparison Operators

Comparison operators are used to compare two values.

Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Example

```
a = 21
b = 10
c = 0
if ( a == b ):
    print ("Line 1 - a is equal to b")
else:
    print ("Line 1 - a is not equal to b")

if ( a != b ):
    print ("Line 2 - a is not equal to b")
else:
    print ("Line 2 - a is equal to b")

if ( a != b ):
    print ("Line 3 - a is not equal to b")
else:
    print ("Line 3 - a is equal to b")

if ( a < b ):
    print ("Line 4 - a is less than b")
else:
    print ("Line 4 - a is not less than b")
```

Output

```
Line 1 - a is not equal to b
Line 2 - a is not equal to b
Line 3 - a is not equal to b
Line 4 - a is not less than b
```

Example -2

```
a = 5;
b = 20;
if ( a <= b ):
    print ("Line 6 - a is either less than or equal to b")
else:
    print ("Line 6 - a is neither less than nor equal to b")
if ( b >= a ):
    print ("Line 7 - b is either greater than or equal to b")
else:
    print ("Line 7 - b is neither greater than nor equal to b")
```

Output

Line 6 - a is either less than or equal to b
Line 7 - b is either greater than or equal to b

4. Logical Operators

Logical operators are used to combine conditional statements.

Operator	Description	Example
and	Returns True if both statements are true	$x < 5$ and $x < 10$
or	Returns True if one of the statements is true	$x < 5$ or $x < 4$
not	Reverse the result, returns False if the result is true	not($x < 5$ and $x < 10$)

Example #1:

```
# Python program to demonstrate
# logical and operator
a = 10
b = 10
c = -10
if a > 0 and b > 0 :
    print("The numbers are greater than 0")

if a > 0 and b > 0 and c > 0:
    print("The numbers are greater than 0")
else :
    print("Atleast one number is not greater than 0")
```

Output:

The numbers are greater than 0
Atleast one number is not greater than 0

Example #2:

```
# Python program to demonstrate
# logical and operator
a = 10
b = 12
c = 0
if a and b and c:
    print("All the numbers have boolean value as True")
else:
    print("Atleast one number has boolean value as False")
```

Output:

Atleast one number has boolean value as False

Example #3:

```
# Python program to demonstrate
# logical not operator
a = 10
if not a:
    print("Boolean value of a is True")

if not (a%3 == 0 or a%5 == 0) :
    print("10 is not divisible by either 3 or 5")
else :
    print("10 is divisible by either 3 or 5")
```

Output:

10 is divisible by either 3 or 5

Explanation

```
# First If Statement
not a  = not True  = False

# Second If Statement
a % 3 = 10 % 3 = 1  ( 1==0 ) False
a % 5 = 10 % 5 = 0  ( 0==0 ) True
not (False or True ) = not ( True ) = False
```


Day-03:

Q.1 Multiple Choice Question

1. Which is an invalid String ?
 - a. 'ABC'
 - b. "ABC"
 - c. ""ABC""
 - d. ""ABC"""
2. Which is an invalid in Python ?
 - a. "This is Python"
 - b. [10 ,20 , "ABC"]
 - c. (10 ,20 , "ABC")
 - d. # 10 ,20 , "ABC" #
3. Which is not true for List Data Type ?
 - a. Items enclosed in the [].
 - b. Items must be of similar type.
 - c. + sign concatenate the list.
 - d. * sign repeats the list.
4. Which is not true for Tuple Data Type ?
 - a. Items enclosed in the [].
 - b. Items can be of any type.
 - c. Items are read only.
 - d. * sign repeats the tuple.
5. Which is an invalid Arithmetic Operator?
 - a. **
 - b. /
 - c. //
 - d. ***

Q.2. What is the output of following?

a) str = 'Hello NIELIT!' b = str + 'Hello' print(b)	b) str = 'Hello NIELIT!' b = str * 2 print(b)
c) str = 'Hello NIELIT!' b = str[2:4] + str[7:9] print(b)	d) str = 'Hello NIELIT!' b = str[2:4] + str[7:9] print(b[2:4])

Q.3 . What is the output of following?

a) <code>x = ['abcd', 786 , 2.23, 'john', 70.2]</code> <code>b = x + 'Hello'</code> <code>print(b)</code>	b) <code>x = ['abcd', 786 , 2.23, 'john', 70.2]</code> <code>b = x*2</code> <code>print(b)</code>
c) <code>x = ['abcd', 786 , 2.23, 'john', 70.2]</code> <code>b = x[1] + x[2]</code> <code>print(b)</code>	d) <code>x = ['abcd', 786 , 2.23, 'john', 70.2]</code> <code>x[1]= 100</code> <code>b = x[1] + x[2]</code> <code>print(b)</code>

Q.4. What is the output of following?

a) <code>x = ('abcd', 786 , 2.23, 'john', 70.2)</code> <code>b = x + 'Hello'</code> <code>print(b)</code>	b) <code>x = ('abcd', 786 , 2.23, 'john', 70.2)</code> <code>b = x*2</code> <code>print(b)</code>
c) <code>x = ('abcd', 786 , 2.23, 'john', 70.2)</code> <code>b = x[1] + x[2]</code> <code>print(b)</code>	d) <code>x = ('abcd', 786 , 2.23, 'john', 70.2)</code> <code>x[1]= 100</code> <code>b = x[1] + x[2]</code> <code>print(b)</code>

Q.5. What is the output of following?

a) <code>x = { 'abcd', 786 , 2.23, 'john', 70.2 }</code> <code>b = x + 'Hello'</code> <code>print(b)</code>	b) <code>x = { 'abcd', 786 , 2.23, 'john', 70.2 }</code> <code>b = x*2</code> <code>print(b)</code>
c) <code>x = { 'abcd', 786 , 2.23, 'john', 70.2 }</code> <code>b = x[1] + x[2]</code> <code>print(b)</code>	d) <code>x = { 'abcd', 786 , 2.23, 'john', 70.2 }</code> <code>x[1]= 100</code> <code>b = x[1] + x[2]</code> <code>print(b)</code>

Q.6. What is the output of following?

a) <code>x = 6/3/2</code> <code>print(x)</code>	b) <code>x=2</code> <code>a = x * -5</code> <code>b = x ** -5</code> <code>print(a)</code> <code>print(b)</code>
c) <code>x = (7 + 3) * 10 / 5</code> <code>print(x)</code>	d) <code>a = 27 // 5</code> <code>b = 27/ 5</code> <code>print(a)</code> <code>print(b)</code>

Q.7. What is the output of following?

a) <code>x = (10 < 5) and ((5 / 0) < 10) print(x)</code>	b) <code>x = (10 > 5) and ((5 / 0) < 10) print(x)</code>
c) <code>x = not (10 < 5) and ((5 / 0) < 10) print(x)</code>	d) <code>x = (10 > 5) or ((5 / 0) < 10) print(x)</code>

Q.8. Evaluate the following Expression. [All the expression will give result T / F]

- a) `5 % 10 + 10 < 50 and 29 >= 29`
- b) `7 ** 2 <= 5 // 9 % 3 or 'bye' < 'Bye'`
- c) `5 % 10 < 10 and -25 > 1 * 8 // 5`
- d) `7 ** 2 // 4 + 5 > 8 or 5 != 6`
- e) `7 / 4 < 6 and 'I am fine' > 'I am not fine'`
- f) `10 + 6 * 2 ** 2 != 9 // 44 - 3 and 29 >= 29 / 9`
- g) `'hello' * 5 > 'hello' or 'bye' < 'Bye'`

Q.9. How does the effect of the following two statement differ ?

- a) `x += x + 10`
- b) `x = x + 10`

Q.10. Identify the Python Keyword from the following list of words:

and class pass if when print not

Q.11. Construct logical expression for representing the following conditions:

- a) marks scored should be greater than 300 and less than 400.
- b) Whether the value of grade is an upper case letter.
- c) The post is engineer and experience is more than four years.

Day-04: Python Operators and Control Structure

Topics to be covered

1. Identity Operators
2. Membership Operators
3. Bitwise Operator
4. Introduction to Control Structure
5. IF Statement

Membership Operators

- Membership operators are used to test if a sequence is presented in an object.

Operator	Description	Example
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

Example-1

```
a = 10
b = 20
list = [1, 2, 3, 4, 5 ]
if ( a in list ):
    print ("Line 1 - a is available in the given list")
else:
    print ("Line 1 - a is not available in the given list")
```

Output

Line 1 - a is not available in the given list

Example-2

```
a = 10
b = 20
list = [1, 2, 3, 4, 5 ]
if ( b not in list ):
    print ("Line 2 - b is not available in the given list")
else:
    print ("Line 2 - b is available in the given list")
```

Output

Line 2 - b is not available in the given list

Identity Operators

Identity operators are used to compare the objects, for

- if they are equal or not.
- If they are equal, means both are same object, with the same memory location.

Operator	Description	Example
is	Returns True if both variables are the same object	x is y
is not	Returns True if both variables are not the same object	x is not y

Example-1

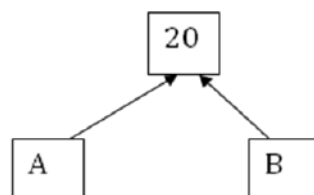
**#In Python, if 2 variable has the same value, then they
#refer the same memory space.**

```
a = 20
b = 20
if ( a is b ):
    print ("Line 1 - a and b have same identity")
else:
    print ("Line 1 - a and b do not have same identity")

print("Id of a= ", id(a)," b= ",id(b))
if ( id(a) == id(b) ):
    print ("Line 2 - a and b have same identity.")
else:
    print ("Line 2 - a and b do not have same identity")
```

Output

```
Line 1 - a and b have same identity
Id of a= 1529084128 b= 1529084128
Line 2 - a and b have same identity.
```



Example -2

**#In Python, if 2 variable has the same value, then they
#refer the same memory space, otherwise different**

```
a = 20
b = 30
if ( a is b ):
    print ("Line 3 - a and b have same identity")
else:
    print ("Line 3 - a and b do not have same identity")
print("Id of a= ", id(a)," b= " ,id(b))
if ( a is not b ):
    print ("Line 4 - a and b do not have same identity")
else:
    print ("Line 4 - a and b have same identity")
```

Output

```
Line 3 - a and b do not have same identity
Id of a= 1529084128 b= 1529084288
Line 4 - a and b do not have same identity
```

Bitwise Operators

- Bitwise operators are used to compare (binary) numbers.

Operator	Name	Description
&	AND	Sets each bit to 1 if both bits are 1
	OR	Sets each bit to 1 if one of two bits is 1
^	XOR	Sets each bit to 1 if only one of two bits is 1
~	NOT	Inverts all the bits
<<	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off
>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off

Bitwise Operator Chart

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

Example-1

```

a = 60          # 60 = 0011 1100
b = 13          # 13 = 0000 1101
c = 0
c = a & b        # 12 = 0000 1100
print ("Line 1 - Value of c is ", c)
c = a | b        # 61 = 0011 1101
print ("Line 2 - Value of c is ", c)
c = a ^ b        # 49 = 0011 0001
print ("Line 3 - Value of c is ", c)
c = ~a          # -61 = 1100 0011
print ("Line 4 - Value of c is ", c)

```

Output

```

Line 1 - Value of c is 12
Line 2 - Value of c is 61
Line 3 - Value of c is 49
Line 4 - Value of c is -61

```

Bitwise &		Bitwise	
A	0011 1100		0011 1100
B	0000 1101		0000 1101
-----		-----	
C	0000 1100		0011 1101
-----		-----	
Bitwise ^		Bitwise ~	
A	0011 1100		0011 1100
B	0000 1101		
-----		-----	
C	0011 0001		1100 0011

Example -2

```
a = 10          # 10 = 0000 1010
c = a << 2      # 40 = 0010 1000
print ("Line 5 - Value of c is ", c)
c = a >> 2      # 2 = 0000 0010
print ("Line 6 - Value of c is ", c)
```

Output

Line 5 - Value of c is 40
Line 6 - Value of c is 2

Python Control Statement

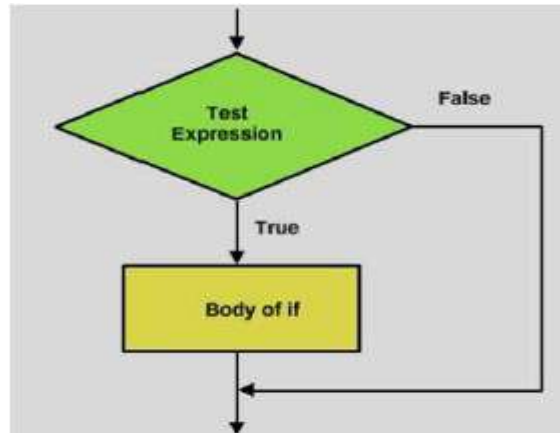
- A control statement modifies the order of statement execution.
- A control statement can cause other program statements
 - to execute multiple times
 - not to execute at all
 - depending on the circumstances.
- Types of Control Statement
 - Branching Statement :- used to select one of the alternative statement.
 - if Statement
 - If....else Statement.
 - iii. If....elif Statement
 - Looping or Iterative Statement :- used to repeat the statement till the condition is true.
 - for loop
 - while loop

IF Statement

- This statement evaluates an expression and directs program execution depending on the result of evaluation.
- It controls the multiple statements through the use of a compound statement or block.
- Block is a group of two or more statements indented at the same level.

IF Syntax

**if expression :
statements**

**Example-1**

```
var1 = 100
if var1:
    print ("1- Got a true expression value")
    print (var1)
var2 = 0
if var2:
    print ("2- Got a true expression value")
    print (var2)
print ("Good bye!")
```

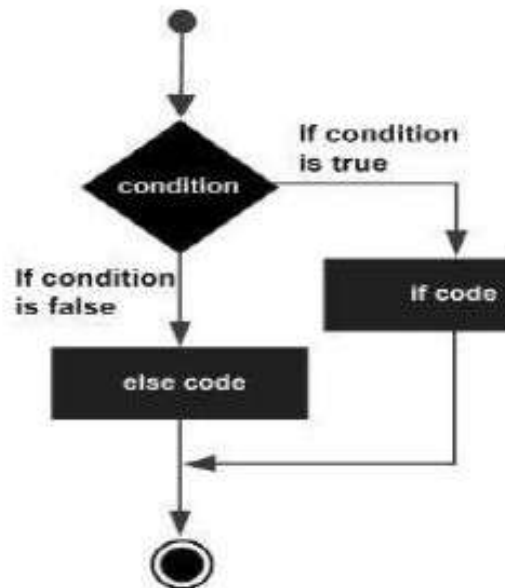
Output

```
1- Got a true expression value
100
Good bye!
```

IF ... Else Statement

- The if..else statement evaluates test expression and will execute body of if only when test condition is True.
- If the condition is False, body of else is executed. Indentation is used to separate the blocks.
- It controls the multiple statements through the use of a compound statement or block.
- Block is a group of two or more statements indented at the same level.
- **Syntax:-**

```
If condition:
    Statement1
else :
    Statement2
```



#Example -1

```
var1 = 100
if var1:
    print ("1- Got a true expression value")
    print (var1)
else:
    print ("1- Got a false expression value")
    print (var1)

print ("Good bye!")
```

Output

```
1- Got a true expression value
100
Good bye!
```

#Example -2

```
var2 = 0
if var2:
    print ("2- Got a true expression value")
    print (var2)
else:
    print ("2 Got a false expression value")
    print (var2)
print ("Good bye!")
```

Output

```
2- Got a false expression value
0
Good bye!
```

If...elif...else Statement

- The elif statement allows you to check multiple expressions for TRUE and execute a block of code as soon as one of the conditions evaluates to TRUE.

- **Syntax**

```
if expression1 :
    statement(s)
elif expression2 :
    statement(s)
elif expression3 :
    statement(s)
else :
    statement(s)
```

Example -1

```
var = 100
if var == 200:
    print ("1 Got a true expression value")
    print (var)
elif var == 150:
    print ("2 Got a true expression value")
    print (var)
elif var == 100:
    print ("3 Got a true expression value")
    print (var)
else:
    print ("4 Got a false expression value")
    print (var)
print "Good bye!"
```

Output

```
3 Got a true expression value
100
Good bye!
```

Day-4

Q.1 Multiple Choice Question

1. Which is an valid Membership Operator?
 - a. not in
 - b. not is
 - c. has
 - d. none
2. Which is an invalid Bitwise Operator ?
 - a. >>
 - b. <<
 - c. >>>
 - d. ^
3. $X=20>>2$ is equal to _____?
 - a. 5
 - b. 10
 - c. 15
 - d. 0
4. $X=5<<2$ is equal to _____ ?
 - a. 5
 - b. 10
 - c. 15
 - d. 20
5. Which is an invalid branching control structure?
 - a. if
 - b. if...else
 - c. if...else...elif...
 - d. if...elif...else...

Q.2 Write a program to display the square and cube of a positive number.

Q.3 Write a program to display the greater of 2 numbers.

Q.4 Write a program to check an entered number is Odd or Even. [**hint** – use % modulus operator to determine the remainder]

Q.5 Write a program to check an entered number is divisible by 7 or not.

Q.6 Write a program to check greatest among three numbers.

Q.7 Write a program to input the number and check it is divisible 3 and 5..

Q.8 Write a program to check a number is positive or negative.

Q.9 Write a program to check a year leap year or not.

Q.10 Write a program to check a three digit number is palindrome or no.

[Hint – use the % operator fragment the digit

$$123 \% 10 = 3$$

$$12 \% 10 = 2$$

$$1 \% 10 = 1$$

Q.11 Write a program to input the cost price and selling price of an item and check for profit or loss. Also calculate it.

Q.12 In an examination, the grades are awarded to the students in 'SCIENCE' according to the average marks obtained in the examination.

Marks	Grades
80% and above	Distinction
60% or more but less than 80%	First Division
45% or more but less than 60%	Second Division
40% or more but less than 45%	Pass
Less than 40%	Promotion not granted

Write a program to input marks in Physics , Chemistry and Biology. Calculate the average marks. Display the average marks and grade obtained.

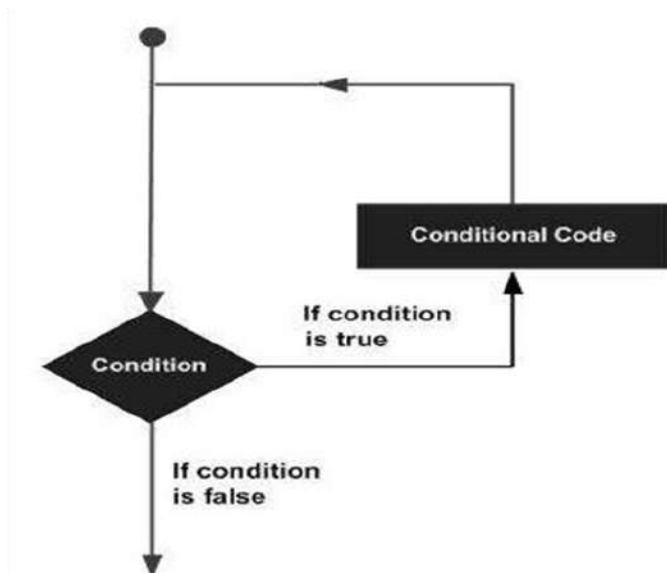
Day-05: Python Control Structure and While Loop

Topics to be covered

1. Loop Control Structure
2. While Loop
3. Break & Continue statement
4. Else and Pass statement
5. Input and Output Statement

Loop Control Structure

- A loop statement allows us to execute a statement or group of statements multiple times.
- Loop Types
 1. **while loop** : Repeats a statement or group of statements while a given condition is TRUE. It tests the condition before executing the loop body.
 2. **for loop** : Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
 3. **nested loops** : You can use one or more loop inside any another while, for or do..while loop.



**Repeats the statements
till the condition is TRUE.**

**Statement is the
indented code.**

While Structure

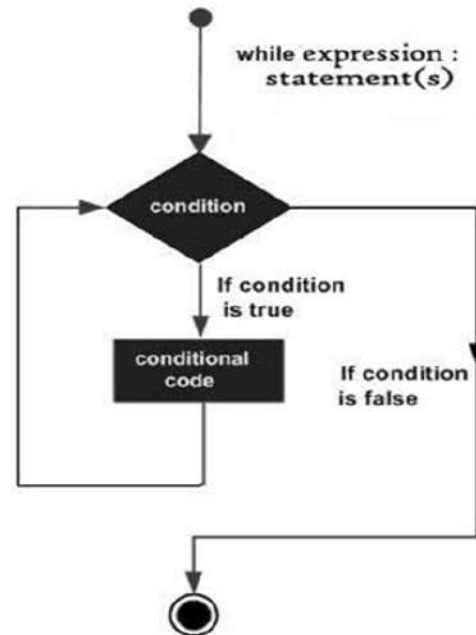
- A while loop statement repeatedly executes a statement as long as a given condition is true.
- Syntax

while expression :
statement(s)

- Statement(s) may be a single statement or a block of statements.
- The condition may be any expression, and **True** is any non-zero value.
- The loop iterates while the condition is **True**.
- Python uses indentation as its method of grouping statements.

**Repeats the statements
till the condition is TRUE.**

**Statement is the
indented code.**



Example-1

Program to print 1 to 10

```

count = 0
while (count < 11) :
    print ('The count is:', count)
    count = count + 1
  
```


Output

The count is: 0
The count is: 1
The count is: 2
The count is: 3
The count is: 4
The count is: 5
The count is: 6
The count is: 7
The count is: 8
The count is: 9
The count is: 10

Example -2**# Program to print 1,3,5,7,9**

```
count = 1
while (count < 10):
    print ( 'The count is:', count)
    count = count + 2
```

Output

The count is: 1
The count is: 3
The count is: 5
The count is: 9

Example -3**# Program to print sum of 1 to 10**

```
count = 1
sum = 0
while (count <= 10) :
    sum=sum + count
    count = count + 1
    print ( 'The sum is:', sum)
```

Output

The sum is: 55

Example-4

Program to factorial of number

```
count = 1
fact = 1
while (count <= 5) :
    fact = fact * count
    count = count + 1
print ('The fact is:', fact)
```

Output

The fact is: 120

Think for

1. Program to print 10 to 1. eg 10 9 8 7 62 1
2. Program to print reverse of 125.

Using else Statement with While Loops

- If the else statement is used with a while loop, the else statement is executed when the condition becomes false.
- Syntax

```
while expression :
    statement(s)
else :
    statement(s)
```

Example-1

Program to print sum of 1 to 10

```
count = 1
sum = 0
while (count <= 10) :
    sum=sum + count
    count = count + 1
else :
    print ( 'The sum is:', sum)
```

Output

The sum is: 55

Example-2**# Program to print Message**

```
count = 0
while count < 5:
    print (count, " - Inside Loop")
    count = count + 1
else:
    print (count, " - Out of Loop")
```

Output

```
0 - Inside Loop
1 - Inside Loop
2 - Inside Loop
3 - Inside Loop
4 - Inside Loop
5 - Out of Loop
```

While Loop and Break Statement

- With the break statement, allows to stop the loop even if the while condition is true.
- Control of the program flows to the statement immediately after the body of the loop.
- If break statement is inside a nested loop (loop inside another loop), break will terminate the innermost loop.
- **Syntax**

```
while expression :
    statement(s)
break
```

```
count=1
while (count <= 10) :
    count = count + 1
    if count == 4 :
        break
    print ( count )
```

This loop will repeats 10 times in normal condition.

Due to **Break statement**, control jumps out of the loop, when count equal to 4.

Example -1

Program to print 1 to 5

```
i = 0
while i < 6:
    i = i + 1
    if i == 3:
        break
    print( i )
```

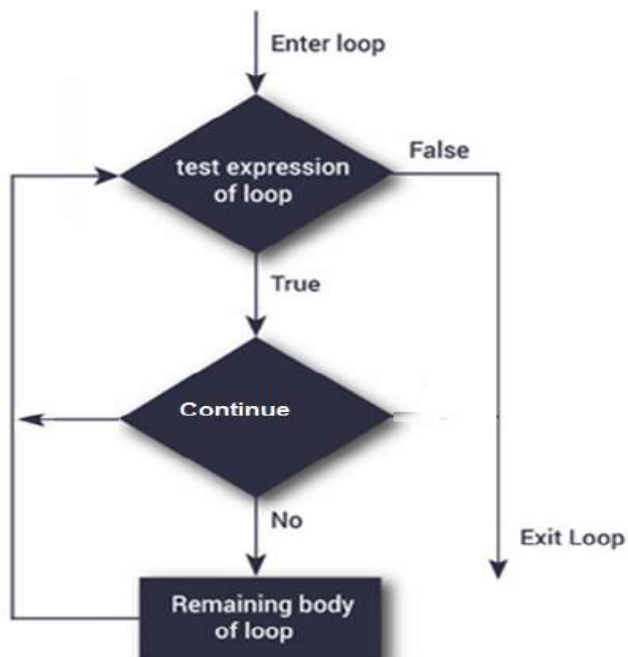
Output

1
2

While Loop and Continue Statement

- With the **continue** statement, we can skip the rest of the code inside a loop for the current iteration only.
- Loop does not terminate but continues on with the next iteration.
- Control of the program remains inside the body of the loop.
- Syntax

while expression :
statement(s)
continue



```
count=1
while (count <= 10) :
    count = count + 1
    if count == 4 :
        continue
    print ( count )
```

This loop will repeats 10 times in normal condition and print 1 to 10.
Due to Continue statement, control skips loop body , when count equal to 4.

#Example -1

Program to print 1 to 5

```
i
while i < 6:
    i = i + 1
    if i == 3:
        continue
    print( i )
```

= 0

Output

```
1
2
4
5
6
```

Input statement

- The **input()** is an in-built function in Python that allows the user to input.
- Data type of the value returned by the **input()** function is string.
- Type Conversion functions are used to convert the string to
 - **int()** - to integer
 - **float()** - to float
- Syntax

x= input("Prompt Message")

Example -1

#Program to use the Input Function

```
x=input("Enter value : ")
print("User Input Value : ", x)
print("Data Type : ", type(x))
```

Output

```
Enter value : 10
User Input Value : 10
Data Type : <class 'str'>
```

Example -2

```
x=input("Enter value : ")
y=input("Enter value : ")
print("Sum = ", x+y)
x=int(x)
y=int(y)
print("After Conversion, Sum = ", x+y)
```

Output

```
Enter value : 10
Enter value : 20
Sum = 1020
After Conversion, Sum = 30
```

Print Statement

- The `print()` is an in-built function in Python that allows the user to display the message or value of variable.
- Syntax
 - **`print("Message" , Variable , [sep , end])`**
- **sep** - It allows to specify the separator between two values.
- **end** - It allows to specify the end of line. Default is New Line(`\n`).
- **sep and end both are optional.**

Example -1

Program to use the SEP keyword

```
x=10
y=20
print("Values are ", x , y)
print("Value are ", x , y , sep="##")
```

Output

```
Values are  10 20
Value are ##10##20
```

Example -2

Program to use the END keyword

```
x=10
y=20
print("Value are ")
print(x)
print(y)
print("Value are ", end="")
print(x, end="")
print(y, end="")
```

Output

```
Value are
10
20
Value are 1020
```

Day-5

Q.1 Multiple Choice Question

1. Which one is not allowed with While Loop?
 - a. else
 - b. break
 - c. continue
 - d. elif

2. Which is true about the Break statement with While Loop?
 - a. It skips the remaining body of the loop.
 - b. It repeats the body of the loop.
 - c. It is performs nothing.
 - d. It transfers the flow outside the body of loop.

3. Which is true about the Continue statement with While Loop?
 - a. It skips the remaining body of the loop, jumps to the beginning, for current iteration.
 - b. It repeats the body of the loop.
 - c. It is performs nothing.
 - d. It transfers the flow outside the body of loop.

4. **input()** function return data type is
 1. int
 2. float
 3. bool
 4. str

5. Which is not true about the **print()** function?
 - a. It is an in-built function.
 - b. It prints the output on screen.
 - c. Only integer variables or messages are allowed.
 - d. It apply the new line character at the end.

Q.2 Write the program to display the first 10 terms of the following series :

- a. 1, 3, 5,.....
- b. 2, 4, 6
- c. 1, 4, 9, 16.....
- d. 1.5, 3.0, 4.5, 6.0
- e. -5, -10, -15, -20

Q.3 Write a program to calculate and display the sum of all odd numbers and even numbers between a range of numbers from m to n where $m < n$. Input m and n.

Q.4 Write a program to print the 10 multiples of any entered number.

Q.5 Write a program to display the sum of 10 natural numbers.

Q.6 Write a program to calculate and display the factorial of an entered number.

Q.7 Write a program to count the vowels in entered string.

Q.8 Write a program to find the average of numbers in list.

X = [10,20,30,40,50]

For Loop Structure

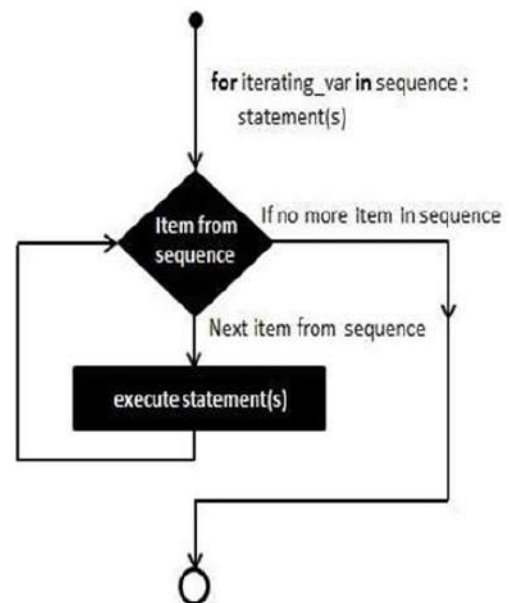
- A for loop is used for iterating the item from the sequence (list, tuple, dictionary, set, string) or range.
- Syntax

**for var in sequence or range :
 statement(s)**

- For loop repeats till the item in the sequence not exhausted.
- Statement(s) may be a single statement or a block of statements.
- Python uses indentation as its method of grouping statements.

Repeats the statements till the items from the sequence not exhausted.

Statements is the indented code.



Example-1

Program to print 1 to 5

```
for i in (1,2,3,4,5) :
    print ( 'The count is:', i )
```

Output

```
The count is: 1
The count is: 2
```

The count is: 3

The count is: 4

The count is: 5

Example-2

Program to print the character from the string

```
for i in 'PYTHON' :
```

```
    print ( i )
```

Output

P

Y

T

H

O

N

Example-3

Program to count the vowels

```
count=0
```

```
for i in "Python Programming":
```

```
    if i in ('a', 'i', 'e', 'o', 'u') :
```

```
        count=count+1
```

```
print ('The count is:', count)
```

Output

The count is: 4

Example -4

Program to count the even no

```
count=0
```

```
for i in (1 ,2 ,3, 4, 5,6):
```

```
    if i%2==0:
```

```
        count=count+1
```

```
print ('The count is:', count)
```

Output

The count is: 3

Think for

1. Program to count the characters other than Vowels.
2. Program to count the numbers divisible by 3.
3. Program to count the numbers divisible by 3 and 5.

Using else Statement with For Loops

- If the else statement is used with a for loop, the else statement is executed when the condition becomes false.
- Syntax

```
for var in Sequence :  
    statement(s)  
else :  
    statement(s)
```

Example-1

```
# Program to count the even.  
count=0  
for i in (1,2,3,4,5):  
    if i % 2==0 :  
        count=count+1  
else :  
    print ( 'The sum is:', sum)
```

Output

The sum is: 15

Example -2**# Program to count the vowels**

```
count=0  
for i in "Python Programming":  
    if i in ('a','i','e','o','u') :  
        count=count+1  
else
```

```
print ('The count is:', count)
```

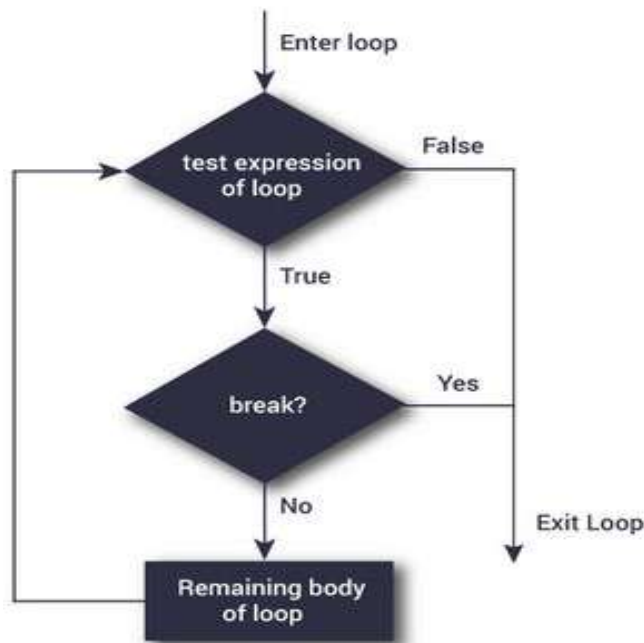
Output

The count is: 4

For Loop and Break Statement

- With the **break** statement, allows to stop the loop even if the sequence in for loop not exhausted.
- Control of the program flows to the statement immediately after the body of the loop.
- If break statement is inside a nested loop (loop inside another loop), break will terminate the innermost loop.
- Syntax

```
for var in sequence :  
    statement(s)  
    break
```



```
count=0
for i in [1,2,3,4,5,6]:
    count=count+1
    if count==3 :
        break
print("Count : ", count)
```

This loop will repeats 6 times in normal condition.

Due to Break statement, control jumps out of the loop, when count equal to 3.

Example-1

Program to use Break

```
count=0
for c in "python":
    if c in ( 'a' , 'e', 'i', 'o', 'u') :
        break
    print(c)
    count=count+1
print("Count ", count)
```

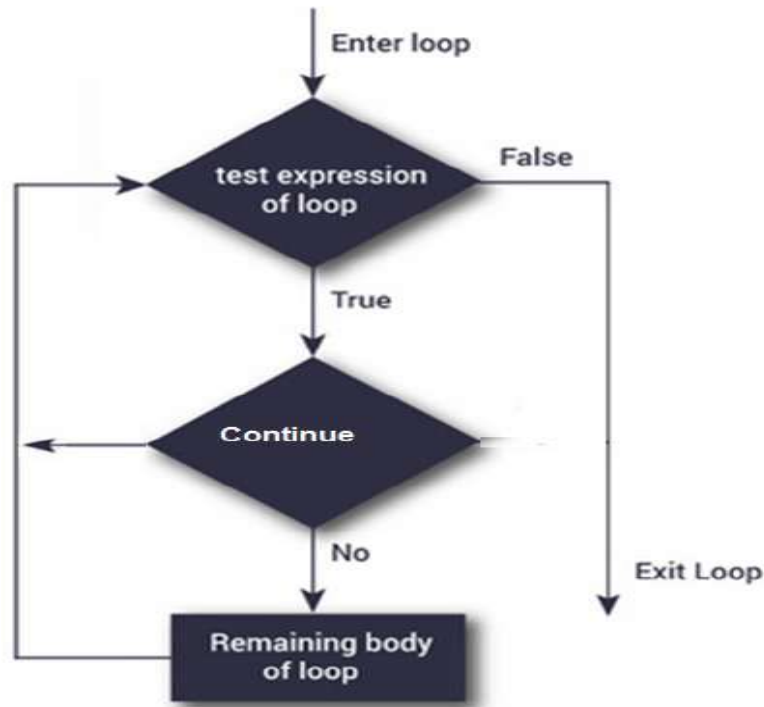
Output

```
p
y
t
h
Count 4
```

For Loop and Continue Statement

- With the **continue** statement, we can skip the rest of the code inside a loop for the current iteration only.
- Loop does not terminate but continues on with the next iteration.
- Control of the program remains inside the body of the loop.
- Syntax

```
for var in sequence :
    statement(s)
    continue
```



```

count=0
for i in [1,2,3,4,5]:
    count=count+1
    if count==3:
        continue
    print("Count :",count)
  
```

This loop will repeats 5 times in normal condition and print 1 to 5.

Due to **Continue statement**, control skips loop body , when count equal to 3.

Example -1

Program to use Continue

```

count=0
for c in "python":
    if c in ('a','e','i','o','u'):
        continue
    print(c)
    count=count+1
    print("Count ", count)
  
```

Output

p
y
t
h
N
Count 5

Range Function

- It is a in-built function.
- The range() function generates the sequence of numbers.
- The sequence starts from 0 by default, and increments by 1 (by default), and ends at a 1 before specified number.
- Syntax

range (start, stop, step)

- start [Optional] - position to start. Default is 0
- stop [Required] - position to end. (**stop – 1**)
- step [Optional] - specifying the increment. Default is 1

```
count=1
while (count <= 10) :
    count = count + 1
    if count == 4 :
        break
    print ( count )
```

Example

- | | |
|--------------------------|-------------------------------------|
| 1. x = range(3, 6) | 3, 4, 5 |
| 2. x = range(3, 20, 2) | 3, 5, 7, 9, 11, 13, 15, 17, 19 |
| 3. x = range(10) | 1, 2, 3, 4, 5, 6, 7, 8, 9 |
| 4. x = range(10, 1, -1) | 10, 9, 8, 7, 6, 5, 4, 3, 2 |
| 5. x = range(-10, -1, 1) | -10, -9, -8, -7, -6, -5, -4, -3, -2 |
| 6. x = range(10, 1, -2) | 10, 8, 6, 4, 2 |

Example -1

#Program to print 1 to 5

```
r=range( 1, 5)
for x in r :
    print( x)
```

Output

```
1
2
3
4
```

Example -2

#Program to print 1 to 10

```
r=range(1, 11 , 1)
sum=0
for x in r :
    sum=sum + x
print("sum ", sum)
```

Output

```
Sum = 55
```

Example-3

#Program to print odd number 1-10

```
r=range( 1, 10, 2)
for x in r :
    print( x)
```

Output

```
1
3
5
7
9
```

Example-4**# Program-4 to factorial of number**

```
count = 1
fact = 1
for x in range(1, 6, 1) :
    fact = fact * x
print ('The fact is:', fact)
```

Output

The fact is: = 120

Enumerate with for loop

- The enumerate() is an in-built function in Python that allows the user to convert the sequences(list, tuple, dictionary) in enumeration list.

- **Example**

```
Months = ["Jan", "Feb", "Mar", "April", "May", "June"]
for i, m in enumerate (Months):
    print (i, m)
```

Output : 0 Jan , 1 Feb, 2 Mar , 3 April, 4 May, 5 June

Example-1**#Program to print odd numbers**

```
r=range(1,10,2)
for i, m in enumerate(r):
    print("Odd Number at ", i, " is ", m)
```

Output

Odd Number at 0 is 1
Odd Number at 1 is 3
Odd Number at 2 is 5
Odd Number at 3 is 7
Odd Number at 4 is 9

Example-2**#Program to print odd numbers**

```
r=range(1,10,2)
for i, m in enumerate(r):
    print("Odd Number at ", i+1, " is ", m)
```

Output

Odd Number at 1 is 1
Odd Number at 2 is 3
Odd Number at 3 is 5
Odd Number at 4 is 7
Odd Number at 5 is 9

Day-6

Q.1 Multiple Choice Question

1. Which one is not allowed with for Loop?
 - a. else
 - b. break
 - c. continue
 - d. elif
2. Which is true about the **Break** statement with for Loop?
 - a. It skips the remaining body of the loop.
 - b. It repeats the body of the loop.
 - c. It is performs nothing.
 - d. It transfers the flow outside the body of loop.
3. Which is true about the **Continue** statement with for Loop?
 - a. It skips the remaining body of the loop, jumps to the beginning.
 - b. It repeats the body of the loop.
 - c. It is performs nothing.
 - d. It transfers the flow outside the body of loop.
4. **for loop not** allows to iterate on is
 - a. int
 - b. list
 - c. tuple
 - d. string
5. Which is not valid for range() function?
 - a. It generates the sequence of numbers upto STOP value.
 - b. START value is optional.
 - c. STEP value is optional.
 - d. Data type of sequence is 'RANGE'.

[USE RANGE and FOR Loop to solve the assignment]

Q.2 Write the program using Range() function to display the first 10 terms of the following series :

- a. 1 , 3, 5,.....19
- b. 4 ,8 , 1240
- c. 1 , 4 , 9 , 16.....100
- d. 2.5 , 5.0 , 7.5 , 9.025.0
- e. -5 , -10 , -15, -20-50
- f. -20, -18,-16, -14 , -12,.....-2

Q.3 Write a program to calculate and display the sum of all odd numbers and even numbers between a range of numbers from m to n where $m < n$. Input m and n. Use the RANGE Function for generating the sequence from 'm' to 'n'.

Q.4 Write a program to print the 10 multiples of any entered number, using Range Function.

Q.5 Write a program to display the sum of 10 natural numbers, using Range Function.

Q.6 Write a program print the items at the odd position the list mentioned below ;

Month = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun']

0 1 2 3 4 5

Output **in this case , items at odd position 1 , 3 & 5 . i.e. Feb Apr Jun**

Use the enumerate Function .

Q.7 Write a program to print the sum of all the numbers at the even position in the sequence generated through range(1,20,2).

Eg range(1,20,3) - 1 3 5 7 9 11 13 15 17 19
 0 1 2 3 4 5 6 7 8 9

Output - sum of items at index 0 , 2 , 4 , 6, 8 = $1 + 5 + 9 + 13 + 17 = 45$

Day-07: Nested Loops and Strings Manipulation

Topics to be covered

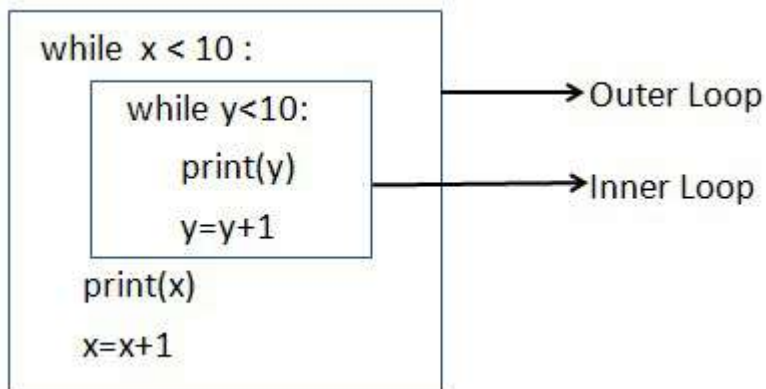
1. Nested While Loop
2. Nested For Loop
3. String
4. String Functions
5. String Formatting Operators

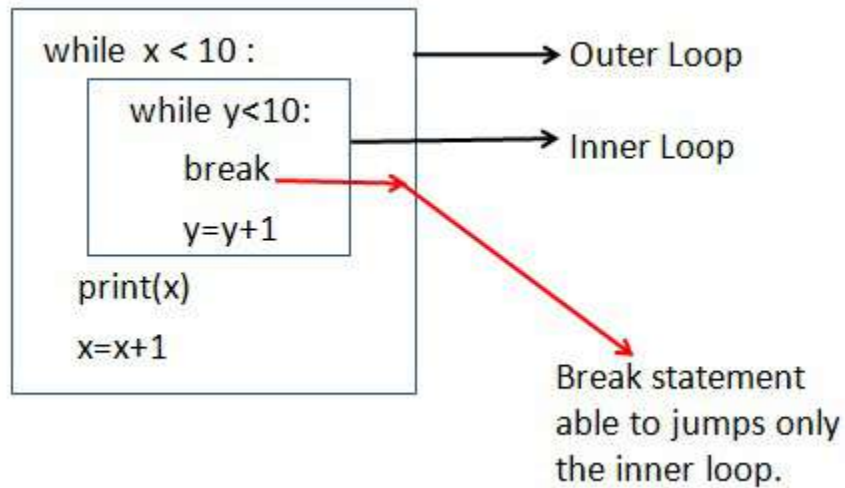
Nested While Loop

- Python allows using one loop inside another while or for loop.
- Syntax for nested While Loop

```
while expression:  
    while expression:  
        statement(s)  
    statement(s)
```

- If break statement is used, it will jumps out of the internal loop only.





Example-1

#Program uses a nested for loop to find #the prime numbers from 2 to 100

```

i = 2
while(i < 100):
    j = 2
    while(j <= (i/2)):
        if not(i % j) : break
        j = j + 1
    if (j > i/2) : print (i, " is prime")
    i = i + 1
  
```

Output

It gives the list of prime number from 2 to 100 .

Example-1

Program uses a nested-for loop to display multiplication tables from 1-10.

```

for i in range(1,11):
    for j in range(1,11):
        k = i*j
        print (k, end=' ')
    print( )
  
```

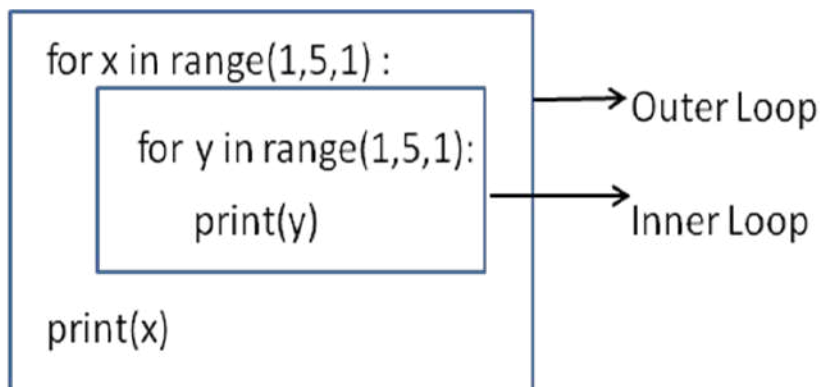
Output

It prints the table of 1 to 10 upto 10 multiples.

Nested For Loop Loop

- Python allows using one loop inside another while or for loop.
- Syntax for nested for Loop


```
for var in sequence :
    for var in sequence :
        statements(s)
        statements(s)
```
- If break statement is used, it will jumps out of the internal loop only.



Example -1

#Program to print the pattern

```
for i in range(1,6):
    for j in range(i) :
        print("*", end=' ')
    print()
```

Output

```
*
**
***
****
*****
```

Explanation

range(1,6) - generate the number 1,2,3,4,5

i=1 range(i) - generate 0

i=2 range(i) - generate 0,1

i=3 range(i) - generate 0,1,2

i=4 range(i) - generate 0,1,2,3

i=5 range(i) - generate 0,1,2,3,4

Example -2

#Program to print the length of words

```
for w in ("Python", "Programming") :  
    len=0  
    for c in w :  
        len=len+1  
    print('Length of Word : ', len)
```

Output

```
Length of Word : 6  
Length of Word : 11
```

Explanation

("Python", "Programming") is a sequence.

Iteration-1

w - "Python"
c - represent each character in word

Iteration-2

w - "Programming"
c - represent each character in word

Strings

- Strings in Python are arrays of bytes representing unicode characters.
- Python does not have a character data type, a single character is simply a string with a length of 1.
- Strings are enclosed characters in quotes. Python treats single quotes the same as double quotes.
- Strings are immutable means that the contents of the string cannot be changed after it is created

Creating String

```
var1 = 'Hello World!'  
var2 = "Python Programming"
```


Access Items

- Square brackets can be used to access elements of the string.
- To access substrings, use the square brackets for slicing along with the index or indices to obtain your substring.

```
var1 = 'Hello World!'
var2 = "Python Programming"

print ("var1[0]: ", var1[0])
print ("var2[1:5]: ", var2[1:5]) print ("var2[2:5]: ", var2[2:6:2])
```

Output –

```
var1[0] : H
var2[1:5]: ytho
```

Negative Indexing

Negative indexing means beginning from the end, -1 refers to the last item, -2 refers to the second last item etc.

String A	S	A	V	E		E	A	R	T	H
Positive Index	0	1	2	3	4	5	6	7	8	9
Negative Index	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

```
var2 = "Python Programming"
print ("var2 last Character: ", var2[-1])
```

Output : g

Range of Indexes (Slicing)

We can specify a range of indexes by specifying where to start and where to end the range.

```
var2 = "Python Programming"
print ("var2[2:5]: ", var2[2:6])
print ("var2[2:5]: ", var2[2:6:2])
```

Output : thon

Output : to

Range of Negative Indexes

Specify negative indexes if you want to start the search from the end of the string.

```
var2 = "Python Programming"
print ("var: ", var2[-4:-1])
print ("var: ", var2[-4:-1:2])
```

Output : min

Output : mn

Updating Strings

We can "update" an existing string by (re)assigning a variable to another string.

```
var2 = "Python Programming"  
var2="Hello "+var2[:6]  
print(var2)
```

Output : Hello Python

Loop Through a String

We can loop through the **String** items by using a **for** loop:

```
var2 = "Python Programming"  
for x in list:  
    print(x)
```

Check if Item String

To check if a certain phrase or character is present in a string, we can use the keywords **in** or **not in**.

```
txt = "The rain in Spain stays mainly in the plain"  
x = "ain" in txt  
print(x)
```

Output True

```
txt = "The rain in Spain stays mainly in the plain"  
x = "ain" not in txt  
print(x)
```

Output False

Length of Set

To determine how many items a **String** has, use the **len()** function.

```
var="Python Programming"  
print(len(var))
```

Escape Character

- To insert characters that are illegal in a string, use an escape character.
- An escape character is a backslash \ followed by the character you want to insert.

The escape character allows you to use double quotes when you normally would not be allowed:

```
txt = "We are the so-called \"Vikings\" from the north."  
print(txt)
```

List of Escape Character

Code	Result
\'	Single Quote
\\	Backslash
\a	Bell or alert
\n	New Line
\r	Carriage Return
\t	Tab
\b	Backspace
\f	Form Feed
\ooo	Octal value
\xhh	Hex value

String Concatenation

To concatenate, or combine, two strings you can use the + operator.

```
a = "Hello"
b = "World"
c = a + b
print(c)                # Hello World
```

String Methods

The **strip()** method removes any whitespace from the beginning or the end

```
a = " Hello, World! "
print(a.strip())        # returns "Hello, World!"
```

The **lower()** method returns the string in lower case:

```
a = "Hello, World!"
print(a.lower())        # returns "hello, world!"
```

The **upper()** method returns the string in upper case:

```
a = "Hello, World!"
print(a.upper())        # returns "HELLO, WORLD!"
```

The **replace()** method replaces a string with another string:

```
a = "Hello, World!"
print(a.replace("H", "J")) # returns "Jello, World!"
```

The **split()** method splits the string into substrings if it finds instances of the separator:

```
a = "Hello, World!"  
print(a.split(","))          # returns ['Hello', ' World!']
```

String Format

- The **format()** method takes the passed arguments, formats them, and places them in the string where the placeholders **{}**.
- We can use index numbers **{0}** to be sure the arguments are placed in the correct placeholders

```
quantity = 3  
itemno = 567  
price = 49.95  
myorder = "I want to pay {2} dollars for {0} pieces of item {1}."  
print(myorder.format(quantity, itemno, price))
```

Output

I want to pay 49.95 dollars for 3 pieces of item 567.

Triple Quotes

- Python's triple quotes come to the rescue by allowing strings to span multiple lines, including NEWLINES, TABs, and any other special characters.
- The syntax for triple quotes consists of three consecutive **single or double** quotes.

```
a = " " "Hello,  
      Python,  
      Programming" " "  
print(a)
```

```
a = ' ' 'Hello,  
      Python,  
      Programming ' ' '  
print(a)
```

String Formatting Operator

- One of Python's coolest features is the string format operator **%**.
- This operator is unique to strings and makes up for the pack of having functions from C's **printf()** family.

```
print ("My name is %s and weight is %d kg!" % ('Zara', 21))
```

Output : My name is Zara and weight is 21 kg!

Sr.No.	Format Symbol & Conversion
1	%c character
2	%s string conversion via str() prior to formatting
3	%i signed decimal integer
4	%d signed decimal integer
5	%u unsigned decimal integer
6	%o octal integer
7	%x hexadecimal integer (lowercase letters)
8	%X hexadecimal integer (UPPERcase letters)
9	%e exponential notation (with lowercase 'e')
10	%E exponential notation (with UPPERcase 'E')
11	%f floating point real number
12	%g the shorter of %f and %e
13	%G the shorter of %f and %E

Example 1.

#Program to check whether the string is a palindrome or not.

```
str=input("Enter the String")
l=len(str)
p=l-1
index=0
while (index<p):
    if(str[index]==str[p]):
        index=index+1
        p=p-1
    else:
        print ("String is not a palidrome" )
        break
else:
    print ("String is a Palidrome" )
```

Example 2.**# Program to count no of 'p' in the string pineapple.**

```
word = 'pineapple'
count = 0
for letter in word:
    if letter == 'p':
        count = count + 1
print(count)
```

Example-3**#Program to check the negative sentence.**

```
str=input("Enter the Sentence:")
if ( 'not' in str):
    print('It is negative sentence')
else:
    print ('It is positive sentence' )
```

Output

Enter the Sentence : Ajay do not like the mobile
It is negative sentence

Example-4**#Program to reverse the string pineapple.**

```
word = 'pineapple'
rev=""
for letter in word:
    rev=letter+ rev

print(' Reverse : ' , rev)
```

Output

Reverse: elppaenip

Day-7

Q.1 Multiple Choice Question

1. Which one is true for nested While Loop?
 - a. Nested loops not allowed with while loop.
 - b. Break jumps out of all the loop.
 - c. Break jumps out of only inner loop.
 - d. Break allowed with only FOR Loop.
2. Which one is true for nested FOR Loop?
 - a. FOR loop allowed on number data type.
 - b. FOR loop allowed on string data type.
 - c. Break not allowed with FOR Loop.
 - d. Nesting not allowed on FOR Loop.
3. Which is true about the **Continue** statement with for Loop?
 - a. It skips the remaining body of the loop, jumps to the beginning.
 - b. It repeats the body of the loop.
 - c. It is performs nothing.
 - d. It transfers the flow outside the body of loop.
4. Which is not a valid format character for strings.
 - a. %f
 - b. %d
 - c. %u
 - d. %t
5. Which function is invalid for string.?
 - a. Strip().
 - b. Lower().
 - c. Split().
 - d. Repeat().

Q.2. Input a string “Green Revolution”. Write a script to print the string in reverse.

`S[::-1]`

Q.3 What will be the output of the following statement? Also justify for answer.

- a) `print('I like Gita\'s pink colour dress')`
- b) `str='Honesty is the best policy'`
`str.replace('o','*')`
- c) `str='Hello World'`
`str.istitle()`
- d) `str="Group Discussion"`
`print (str.lstrip("Gro"))`

Q4 . Consider the string `str="Global Warming"`

Write statements in python to implement the following

- a) To display the last four characters.
- b) To display the substring starting from index 4 and ending at index 8.
- c) To check whether string has alphanumeric characters or not.
- d) To trim the last four characters from the string.
- e) To trim the first four characters from the string.
- f) To display the starting index for the substring "Wa".
- g) To change the case of the given string.
- h) To check if the string is in title case.
- i) To replace all the occurrences of letter 'a' in the string with '*'.

Q.5 Count all lower case, upper case, digits, and special symbols from a given string

Given:

```
str1 = "P@#yn26at^&i5ve"
```

Q.6 Program to calculate the sum and average of the numbers in the string entered in a single line.

[Hint : **split the string with delimiter – space**]

```
X=input('Enter the numbers')
```

For this user has entered - 20 10 30 40 50 6 8 12

Output

```
sum = 176  
average = 22
```

Q.7 Program to count the frequency of vowels in the string.

```
String = "Python is a Programming Language"
```


Q.8 Program to count the frequency of SearchString words in the MainString.

MainString = "Ajay Vijay Sanjay Ajay Ajit Vijay Vikas Rakesh"

SearchString = "Vijay Rakesh"

Output –

Vijay = 2

Rakesh = 1

Q.9 Program to count the words starting with 'A' character in the MainString.

MainString = "Ajay Vijay Sanjay Ajay Ajit Vijay Vikas Rakesh"

Output Count of words starting with 'A' = 3

Python Collections (Arrays)

There are four collection data types in the Python programming language:

- **List** is a collection which is ordered and changeable. Allows duplicate members.
- **Tuple** is a collection which is ordered and unchangeable. Allows duplicate members.
- **Set** is a collection which is unordered and unindexed. No duplicate members.
- **Dictionary** is a collection which is unordered, changeable and indexed. No duplicate members.

Lists

- A list is a collection which is ordered and changeable.
- The list is a datatype available in Python which can be written as a list of comma-separated values (items) between square brackets.
- Items in a list need not be of the same type.

Creating List

```
list1 = ['physics', 'chemistry', 1997, 2000];  
list2 = [1, 2, 3, 4, 5 ];  
list3 = ["a", "b", "c", "d"]
```

Access Items

To access values in lists, use the square brackets for slicing along with the index or indices to obtain value available at that index.

```
list1 = ['physics', 'chemistry', 1997, 2000]  
list2 = [1, 2, 3, 4, 5, 6, 7 ]  
print ("list1[0]: ", list1[0])  
print ("list2[1:5]: ", list2[1:5])  
print ("list1[3]: ", list1[-1])
```

Output–

```
list1[0]: physics  
list2[1:5]: [2, 3, 4, 5]  
list1[3]: 2000
```

Negative Indexing

Negative indexing means beginning from the end, `-1` refers to the last item, `-2` refers to the second last item etc. eg `List[-1]`

Range of Indexes

- You can specify a range of indexes by specifying where to start and where to end the range.
- **List[2:5]** - The search will start at index 2 (included) and end at index 5 (not included).
- Remember that the first item has index 0.

- **List[:5]** - By leaving out the start value, the range will start at the first item:

Range of Negative Indexes

Specify negative indexes if you want to start the search from the end of the list.

List[-4:-1]

Updating Lists

- You can update single or multiple elements of lists by giving the slice on the left-hand side of the assignment operator, and
- To add an item to the end of the list, use the **append()** method.
- To add an item at the specified index, use the **insert()** method.

Example

```
list = ['physics', 'chemistry', 1997, 2000]
print ("Value available at index 2 : ", list[2])
list[2] = 2001
print ("New value available at index 2 : ", list[2])
list.append('Maths')
print("New value available at index 2 : ", list)
```

Output

```
New value available at index 2 : 2001
New value available at index 2 :
['physics', 'chemistry', 2001, 2000, 'Maths']
New value available at index 2 :
['physics', 'Hindi', 'chemistry', 2001, 2000, 'Maths']
```

Loop Through a List

You can loop through the list items by using a for loop:

```
list = ["apple", "banana", "cherry"]
for x in list:
    print(x)
```

Check if Item Exists

To determine if a specified item is present in a list use the **"in"** keyword:

```
list = ["apple", "banana", "cherry"]
if "apple" in list:
    print("Yes, 'apple' is in the fruits list")
```

List Length

To determine how many items a list has, use the **len()** function:

```
print(len(list))
```

Removing Item from the List

- The **remove()** method removes the specified item.

```
list = ["apple", "banana", "cherry"]
list.remove("banana")
```

- The **pop()** method removes the specified index, (or the last item if index is not specified):

```
list.pop()      removes the "cherry"
list.pop(1)     removes the "banana"
```

- The **del** keyword removes the specified index or the complete list.

```
del list[0]
del list
```

- The **clear()** method empties the list

```
list.clear()
```

Basic List Operations

Python Expression	Results	Description
<code>len([1, 2, 3])</code>	3	Length
<code>[1, 2, 3] + [4, 5, 6]</code>	<code>[1, 2, 3, 4, 5, 6]</code>	Concatenation
<code>['Hi!'] * 4</code>	<code>['Hi!', 'Hi!', 'Hi!', 'Hi!']</code>	Repetition
<code>3 in [1, 2, 3]</code>	True	Membership
<code>for x in [1,2,3] : print (x,end = ' ')</code>	1 2 3	Iteration

Matrix implementation using list

We can implement matrix operation using list. Matrix operation can be implemented using nested list. List inside another list is called nested list.

Matrix creation

Program -1

```
# A basic code for matrix input from user
```

```
R = int(input("Enter the number of rows:"))
C = int(input("Enter the number of columns:"))

# Initialize matrix
matrix = [ ]
print("Enter the entries rowwise:")

# For user input
for i in range(R):      # A for loop for row entries
    a = [ ]
    for j in range(C):  # A for loop for column entries
        a.append(int(input()))
    matrix.append(a)

# For printing the matrix
for i in range(R):
    for j in range(C):
        print(matrix[i][j], end = " ")
    print()
```

Tuple

- A tuple is a collection which is ordered and unchangeable (immutable).
- The tuple is a **datatype** in Python which allows comma-separated values (items). Optionally, you can put these comma-separated values between parentheses also.
- Items in a tuple need not be of the same type.
- Example
 - t1 = ('physics', 'chemistry', 1997, 2000)
 - t2 = (1, 2, 3, 4, 5)
 - t3 = "a", "b", "c", "d"
- Tuple() – convert the object to TupleType.
 - t1 = tuple("Python") ---- ('P', 'y', 't', 'h', 'o', 'n')
 - t2=tuple([2,4,5]) ---- (2, 4, 5)

To create a tuple with only one item, you have to add a comma after the item, otherwise Python will not recognize it as a tuple.

```
thistuple = "apple", "mango"
print(type(thistuple))

thistuple = ( )
print(type(thistuple))

# a tuple with one item
```

```
thistuple = ("apple",)  
print(type(thistuple))
```

#NOT a tuple

```
thistuple = ("apple")  
print(type(thistuple))
```

Output

```
<class 'tuple'>  
<class 'tuple'>  
<class 'tuple'>  
<class 'str'>
```

Access Items – To access item, we use the square brackets for slicing along with the index or indices.

- Example

```
tup1 = ['physics', 'chemistry', 1997, 2000]  
tup2 = [1, 2, 3, 4, 5, 6, 7]  
print ("tup1[0]: ", tup1[0])           ---- tup1[0]: physics  
print ("tup2[1:5]: ", tup2[1:5])      ---- tup2[1:5]: (2, 3, 4, 5)
```

Negative Index- It means from the end, -1 refers to the last item, -2 refers to second last item.

```
tup1 = ['physics', 'chemistry', 1997, 2000]  
print ("tup1[3]: ", tup1[-1])         ----- tup1[3]: 2000
```

Operations on Tuple

1. Range of Indexes (Slicing)

```
tup1 = ('physics', 'chemistry', 1997, 2000)  
print ("tup1[1:5]: ", tup1[1:3])      ---- ('chemistry', 1997 )  
print ("tup1[1:5:2]: ", tup1[1:3:2])  ---- ('chemistry', 2000)
```

2. Negative Indexes

```
tup1 = ( 'physics', 'chemistry', 1997, 2000 )  
print ("tup1[1:5]: ", tup1[-3:-1])    ----- ('chemistry', 1997)  
print ("tup1[1:5:2]: ", tup1[-3:-1:2]) ----- ('chemistry')
```

3. Loop Through a Tuple

```
tup1 = ('physics', 'chemistry', 1997, 2000)  
for x in tup1:  
    print(x)
```

4. Updating Tuple

```
tup1 = (12, 34.56)
```

```
tup2 = ('abc', 'xyz')
```

```
# Following action is not valid for tuples
```

```
# tup1[0] = 100
```

```
# create a new tuple
```

```
tup3 = tup1 + tup2
```

```
print (tup3)
```

```
# add new value in tuple
```

```
tup1 = tup1 + (60, )
```

```
print (tup1)
```

Output

```
(12, 34.56, 'abc', 'xyz')
```

```
(12, 34.56, 60)
```

5. Method to update the values in Tuple

```
x = ("apple", "banana", "cherry" )
```

```
y = list(x)
```

```
y[1] = "kiwi"
```

```
x = tuple(y)
```

```
print(x)
```

Output

```
("apple", "kiwi", "cherry")
```

6. Check if Item in Tuple

```
tup= ( "apple", "banana", "cherry" )
```

```
if "apple" in tup :
```

```
print("Yes, 'apple' is in the fruits list")
```

7. Length of Tuple

```
tup= ("apple", "banana", "cherry")
```

```
print( len(tup) )
```

8. Tuple Concatenation

```
a = (1, 2, 3 )
```

```
b = ( 4, 5, 6 )
```

```
c = a + b
```

```
print(c)           # (1, 2, 3, 4, 5, 6 )
```

9. Repeat the Tuple with *

```
a = ( 1, 2, 3 )
```

```
print(a * 3)       # (1, 2, 3, 1, 2, 3, 1, 2, 3 )
```

10. del - Tuples can be deleted only. Removing item not allowed.

```
a = (1, 2, 3 )
```

```
del a
print( a )          # Error
```

11. index(item) - Gives the index of item.

```
a = (1, 2, 3 )
print(a.index(2) )    # 1
```

12. max() - maximum from the tuple.

```
a = ( 1, 2, 3 )
print(max(a))         # 3
```

Basic Tuple Operations

Python Expression	Results	Description
len((1, 2, 3))	3	Length
(1, 2, 3) + (4, 5, 6)	(1, 2, 3, 4, 5, 6)	Concatenation
('Hi!') * 4	['Hi!', 'Hi!', 'Hi!', 'Hi!']	Repetition
3 in (1, 2, 3)	True	Membership
for x in (1,2,3) : print (x,end = ' ')	1 2 3	Iteration

Example-1

#Write a program to input 'n' numbers and # store it in tuple.

```
t=tuple()
n=input("Enter any number : ")
n=int(n)
print (" enter all numbers one after other : " )
for i in range(n):
    a=input("enter number : ")
    t=t+(a,)
print ("output is" )
print ( t )
```

Output

```
Enter any number : 5
enter all numbers one after other
enter number : 10
enter number : 20
enter number : 30
enter number : 40
enter number : 50

output is
('10', '20', '30', '40', '50')
```

SET

- A Set is a collection which is unordered and unindexed.
- The Set is a **datatype** which allows a list of comma-separated values (items) between square brackets.
- Items in a list need not be of the same type.
- Example
 - `set1 = { 'physics', 'chemistry', 1997, 2000 }`
 - `set2 = { 1, 2, 3, 4, 5 }`
 - `set3 = { "a", "b", "c", "d" }`
- `set()` – convert the object to Set Type.
 - `s1 = set("Python")` ---- `{'P', 'y', 't', 'h', 'o', 'n' }`
 - `s2 = set((2,4,5))` ---- `{ 2, 4, 5 }`
- The sets allows the mathematical operations like union, intersection, difference and complement etc
- Access Items – Individual items not accessible in SET. Access all the items together.
- **Example**

```
set1 = { 'physics', 'chemistry', 1997, 2000 }
set2 = { 1, 2, 3, 4, 5, 6, 7 }
print ("set1: ", set1)
print ("set1[1]: ", set1[1])      # Index not allowed in Set
print ("tset2[3]: ", set2[-1])    # Negative Index not allowed in Set
```
- Negative Indexing , Slicing , Updating , + and * not allowed on SETs
 - `# set1[0] = 100`
 - `# set3 = set1 + set2`

Basic Operations

1. `add()` - Add One Item in SET.

```
s = { 10, 20, 30, 40 }
s = s.add(60)
print( s)                # { 10, 20, 30 ,40 , 60 }
```

2. `update(list)` - Add more the one Item in SET.

```
s = { 10, 20 }
s = s.update( [50,60] )
print( s)                # { 10, 20, 50 , 60 }
s = s.update( 'ABC' )
print( s)                # { 10, 20, 50 , 60 , 'A', 'B', 'C' }
```

3. Loop Through a SET

```
s1 = { 'physics', 'chemistry', 1997, 2000 }
for x in s1
print(x)
```

4. Length of SET

```
s= { "apple", "banana", "cherry" }
print(len(s))
```

5. Check if Item in SET

```
s= { "apple", "banana", "cherry" }  
if "apple" in s:  
    print("Yes, 'apple' is in the fruits SET")
```

6. remove() - remove the item.

```
a = { 1, 2, 3 }  
print(a.remove(2 ))          # [1, 3]
```

7. pop() - remove the last item in unordered SET .

```
a = { 1, 2, 3 }  
print(a.pop( ))              # [1, 3]
```

8. clear() and del - removes all the items.

```
a = [1, 2, 3]  
print(a.clear( ))            # None
```

9. max() - maximum from the SET.

```
a = { 1, 2, 3 }  
print(max(a))                # 3
```

SET Operation**1. Union of Sets**

- The union operation produces a new set containing all the distinct elements from both the sets.
- **union()** or **|** used for union operation on two set.

Example

```
DaysA = {"Mon", "Tue", "Wed" }  
DaysB = { "Wed", "Thu", "Fri", "Sat", "Sun" }  
D1 = DaysA | DaysB  
D2 = DaysA.union( DaysB )  
print(D1)          ----- {'Thu', 'Sat', 'Tue', 'Sun', 'Mon', 'Fri', 'Wed'}  
print(D2)          ----- {'Sat' , 'Thu', 'Tue', 'Sun', 'Mon', 'Wed', 'Fri'}
```

2. Intersection of Sets

- The intersection operation produces a new set containing only the common elements from both the sets.
- **intersection()** or **&** used for intersection operation on two set.

Example

```
DaysA = {"Mon", "Tue", "Wed" }
```

```
DaysB = { "Wed", "Thu", "Fri", "Sat", "Sun" }  
D1 = DaysA & DaysB  
D2 = DaysA.intersection( DaysB )  
print(D1)          ----- { 'Wed' }  
print(D2)          ----- { 'Wed' }
```

3. Difference of Sets

- The difference operation produces a new set containing only the elements from the first set and none from the second set.
- **difference()** or **-** used for difference operation on two set.

Example

```
DaysA = { "Mon", "Tue", "Wed" }  
DaysB = { "Wed", "Thu", "Fri", "Sat", "Sun" }  
D1 = DaysA - DaysB  
D2 = DaysA.difference( DaysB )  
print(D1)          ----- { 'Mon', 'Wed' }  
print(D2)          ----- { 'Mon', 'Wed' }
```

4. Disjoint of Sets

- This method generated TRUE if none of the items are present in both sets.
- **isdisjoint()** used for disjoint on two set.

Example

```
DaysA = { "Mon", "Tue", "Wed" }  
DaysB = { "Wed", "Thu", "Fri", "Sat", "Sun" }  
D1 = DaysA.isdisjoint ( DaysB )  
print(D1)          ----- False
```

5. Compare Set

- The **issubset()** method returns True if all items in the set exists in the specified set, otherwise it returns False. **<=** can also be used for subset.
- The **issuperset()** method returns True if all items in the specified set exists in the original set, otherwise it returns False. **>=** can also be used for superset

Example

```
DaysA = { "Mon", "Tue", "Wed" }  
DaysB = { "Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun" }  
Subset = DaysA <= DaysB  
Superset = DaysB >= DaysA  
print(Subset)          ----- True  
print(Superset)        ----- True
```

6. Symmetric difference

- This method returns a set that contains all items from both set, but not the items that are present in both sets.

Example

```
DaysA = {"Mon", "Tue", "Wed" }  
DaysB = {"Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun" }  
D=DaysA.symmetric_difference(DaysB)  
print(D)
```

Output

```
{'Sat', 'Fri', 'Sun', 'Thu'}
```

Day -8

Q.1 Multiple Choice Question

- Which of the following statements is used to create an empty set?
 - `{ }`
 - `[]`
 - `()`
 - `set()`
- Which of the following lines of code will result in an error?
 - `s={abs}`
 - `s={4, 'abc', (1,2)}`
 - `s={2, 2.2, 3, 'xyz'}`
 - `s={san}`
- What is the output of the line of code shown below, if `s1= {1, 2, 3}`? `s1.issubset(s1)`
 - True
 - Error
 - No Output
 - False
- Suppose `t = (1, 2, 4, 3)`, which of the following is incorrect?
 - `print(t[3])`
 - `t[3] = 45`
 - `print(max(t))`
 - `print(len(t))`
- If `a=(1,2,3,4)`, `a[1:-1]` is _____
 - Error, tuple slicing doesn't exist
 - `[2,3]`
 - `(2,3,4)`
 - `(2,3)`

Q.2 What is the output of the following code:

a) <code>print (type ([1,2]))</code>	b) <code>a= [1, 2, 3, None, (), []]</code> <code>print(a)</code>
c) <code>A= [1,2,3,4,5]</code> <code>L=len(A)</code> <code>S=0</code> <code>for I in range(1,L,2):</code> <code> S+=A[I]</code> <code>print("Sum=",S)</code>	d) <code>t=tuple()</code> <code>t = t +("PYTHON",)</code> <code>print(t)</code> <code>print (len(t))</code> <code>t1=(10,20,30)</code> <code>print (len(t1))</code>

Q.3 For each of the expression below, specify its output. If it generates error, write error:

1. ListA= [1, 4, 3, 0]
2. ListB= ['x', 'z', 't', 'q']
3. ListA.sort ()
4. print(ListA)
5. ListA.insert (0, 100)
6. ListA.remove (3)
7. ListA.append (7)
8. ListC=ListA+ListB
9. ListB.pop ()
10. ListA.extend ([4, 1, 6, 3])

Q.4 Create a list that contains the names of 5 students of the class.

1. Print the list
2. Ask the user to input one name and append it to the list
3. Print the list
4. Ask user to input a number. Print the name that has the number as index (Generate error message if the number provided is more than last index value).
5. Add “Kamal” and “Sanjana” at the beginning of the list by using “+”.
6. Print the list
7. Ask the user to type a name. Check whether that name is in the list. If exist, delete the name, otherwise append it at the end of the list.
8. Create a copy of the list in reverse order .
9. Print the original list and the reversed list.
10. Remove the last element of the list.

Q.5 Write the output from the following code:

```
x = {"a", "b", "c"}
y = {"c", "d", "e"}
z = {"f", "g", "c"}

result = x.intersection(y, z)
print (result)
```

Q.6 Write a program to input ‘n’ numbers in two sets and merge the Set in the following manner.

Example

```
Input : T1 = {10,30,50 }
        T2= {20,40,60}
Output :
        T3= {10,20,30,40,50,60}
```

Q.7 Let $A = \{ "a", "b", "d", "e" \}$, $B = \{ "b", "c", "e", "f" \}$ and $C = \{ "d", "e", "f", "g" \}$

(i) Verify $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$

(ii) Verify $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$

Q.8 Given sets are

set1 = { 10, 20, 30, 40, 50 }

set2 = { 30, 40, 50, 60, 70 }

- a) Find a set of identical items from the two sets.
- b) Find a new set with all items from both sets by removing duplicates
- c) Find a set of all elements in either A or B, but not both
- d) Determines whether or not the following two sets have any elements in common. If yes display the common elements

Dictionary

- A dictionary is a collection which is unordered, changeable and indexed.
- Dictionaries are written with curly brackets, and they have keys and values.
- Each key is separated from its value by a colon (:)
- **Keys** are unique and **Values** may not be.
- The values of a dictionary can be of any immutable data type, like strings, numbers, or tuples.
- **dictionary()** – convert the object to dictionary Type.

Creating Dictionary

```
dict1 = {  
    "brand" : "Ford",  
    "model" : "Mustang",  
    "year" : 1964  
}  
  
dict2 = {'Name' : 'Zara', 'Age' : 7, 'Class' : 'First' }  
  
dict3 = { }      #Empty Dictionary
```

Access Items

- We can access the items of dictionary by referring to its key name, inside square brackets.
- **get()** method will be used to get the value of key.

```
dict1 = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}  
print ( dict1 )  
print ( dict1['Age'] )  
x= dict1['Class']  
print ( x )  
x= dict1.get('Name')  
print ( x )
```

Output–

```
{'Name': 'Zara', 'Age': 7, 'Class': 'First'}  
7  
First  
Zara
```

Updating Dictionary

We can update a dictionary by adding a new entry or a key-value pair, modifying an existing entry, or deleting an existing entry

```
dict1 = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}  
dict1['Age'] = 10
```



```
print ( dict1['Age'] )
```

Output

```
{'Name': 'Zara', 'Age': 10, 'Class': 'First'}
```

Adding new item in Dictionary

Adding an item to the dictionary is done by using a new index key and assigning a value to it.

```
dict1 = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}  
print ( dict1 )  
dict1['Height'] = 4  
print ( dict1 )
```

Output

```
{'Name': 'Zara', 'Age': 7, 'Class': 'First'}  
{'Name': 'Zara', 'Age': 7, 'Class': 'First', 'Height': 4}
```

Loop Through a Dictionary

- When looping through a dictionary, the return values are the **keys** of the dictionary, but there are methods to return the **values** as well.
- use the **keys()** function to return keys of a dictionary.
- use the **values()** function to return values of a dictionary.
- Loop through both *keys* and *values*, by using the **items()** function

```
dict1 = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}  
for x in dict1:          #Print all key names in the dictionary, one by one  
    print(x)  
  
for x in dict1:          #Print all values in the dictionary, one by one  
    print(dict1[x])
```

```
for x in dict1.values( ): #Print all values in the dictionary, using the values() function  
    print(x)  
  
for x , y in dict1.items( ): #Print both keys and values, by using the items() function  
    print(x , y)
```

Output

```
Name  
Age  
Class  
  
Zara
```

7

First

Zara

7

First

Name Zara

Age 7

Class First

Check if Item Dictionary

To determine if a specified item is present in a Dictionary use the “**in**” keyword:

```
dict1 = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}  
if "Age" in dict1:  
    print("Yes, 'Age' is the key in the Dictionary")
```

Length of Dictionary

To determine how many items (key-value pairs) a **Dictionary** has, use the **len()** function.
e.g. `print(len(dict1))`

Removing Item from the Dictionary

- The `pop()` method removes the item with the specified key name.

```
dict1 = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}  
dict1.pop('Name')  
print(dict1)
```

- The `popitem()` method removes the last inserted item.

```
dict1 = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}  
dict1.popitem()  
print(dict1)
```

- The `del` keyword removes the item with the specified key name.

```
dict1 = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}  
dict1.del('Name')  
print(dict1)
```

- The `del` keyword can also delete the dictionary completely.

```
dict1 = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}  
del dict1
```

```
print(dict1)
```

- The `clear()` method empties the dictionary.

```
dict1 = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}  
dict1.del('Name')  
print(dict1)
```

Copy a Dictionary

- We cannot copy a dictionary simply by typing `dict2 = dict1`, because: `dict2` will only be a reference to `dict1`, and changes made in `dict1` will automatically also be made in `dict2`.
- To make a copy, one way is to use the built-in Dictionary method **`copy()`**.
- Another way to make a copy is to use the built-in method or constructor **`dict ()`**.

```
dict1 = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}  
dict2= dict1.copy( )  
print(dict2)  
  
dict3=dict(dict1)  
print(dict2)
```

Nested Dictionaries

A dictionary can also contain many dictionaries, this is called nested dictionaries.

```
myfamily = {  
    "child1" : { "name" : "Emil", "year" : 2004 },  
    "child2" : { "name" : "Tobias", "year": 2007 },  
    "child3" : { "name" : "Linus", "year" : 2011 }  
}  
print(myfamily)
```

Output

```
{'child1': {'name': 'Emil', 'year': 2004}, 'child2': {'name': 'Tobias', 'year': 2007},  
'child3': {'name': 'Linus', 'year': 2011}}
```

Function

A function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for application and a high degree of code reusing.

Functions

- A function is a named, independent section of Python code that performs a specific task and optionally returns a value to the calling program.

- A function is named. Each function has a unique name.
- By using the name in another part of the program, one can execute the statements contained in the function. **This is known as calling the function.**
- A function can be called from within any other function.
- A function is independent.
- A function can perform its task without interference from or interfering with other parts of the program.

Types of functions

- 1) **Predefined standard library functions** – such as input(), list(), str(), format() etc – These are the functions which already have a definition library files, so we just call them whenever there is a need to use them.
- 2) **User Defined functions** – The functions that we create in a program are known as user defined functions.

Need functions in C

Functions are used because of following reasons –

- 1) To improve the readability of code.
- 2) Improves the reusability of the code, same function can be used in any program rather than writing the same code from scratch.
- 3) Debugging of the code be easier if you use functions, as errors are easy to be traced.
- 4) Reduces the size of the code, duplicate set of statements are replaced by function calls.

Syntax of a function

```
def function_name( parameters ):  
    "function_docstring"  
    function_suite  
    return [expression]
```

Built-in Python Functions

SNo	Function	Description
1.	input()	Allowing user input
2.	print()	Prints to the standard output device
3.	int()	Returns an integer number
4.	float()	Returns a floating point number
5.	list()	Returns a list

6.	dict()	Returns a dictionary (Array)
7.	set()	Returns a new set object
8.	str()	Returns a string object
9.	tuple()	Returns a tuple
10.	type()	Returns the type of an object
11.	len()	Returns the length of an object
12.	format()	Formats a specified value
13.	abs()	Returns the absolute value of a number
14.	eval()	Evaluates and executes an expression
15.	round()	Rounds a numbers
16.	max()	Returns the largest item in an iterable
17.	min()	Returns the smallest item in an iterable
18.	oct()	Converts a number into an octal
19.	pow()	Returns the value of x to the power of y
20.	range()	Returns a sequence of numbers, starting from 0 and increments by 1 (by default)

1. **eval() Function** - evaluates the expression.

```
x= 'print(55)'
eval(x)
x= eval( '10+5' )
print(x)
```

Output-

```
55
5
```

2. **type() Function** - the type of the specified object.

```
a = ('apple', 'banana', 'cherry')
b = "Hello World"
c = 33
print(type(a) )      ---- <class 'tuple'>
print(type(b))       ---- <class 'str'>
print(type(c))       ---- <class 'int'>
```

3. **range()** Function - returns a sequence of numbers.

#sequence of numbers from 0 to 5

```
x = range(6)
for n in x :
    print(n)
```

#sequence of numbers from 3 to 5

```
x = range(3, 6)
for n in x :
    print(n)
```

#sequence of num from 3 to 19, increment by 2

```
x = range(3, 20, 2)
for n in x :
    print(n)
```

4. **format()** Function

The **format()** function formats a specified value into a specified format

format(value, format)

Parameter	Description
<i>value</i>	A value of any format
<i>format</i>	<p>The format you want to format the value into. Legal values:</p> <p>'<' - Left aligns the result (within the available space) '>' - Right aligns the result (within the available space) '^' - Center aligns the result (within the available space) '=' - Places the sign to the left most position '+' - Use a plus sign to indicate if the result is positive or negative '-' - Use a minus sign for negative values only ' ' - Use a leading space for positive numbers ',' - Use a comma as a thousand separator '_' - Use an underscore as a thousand separator 'b' - Binary format 'c' - Converts the value into the corresponding Unicode character 'd' - Decimal format 'e' - Scientific format, with a lower case e 'E' - Scientific format, with an upper case E 'f' - Fix point number format 'F' - Fix point number format, upper case 'g' - General format 'G' - General format (using a upper case E for scientific</p>

	notations) 'o' - Octal format 'x' - Hex format, lower case 'X' - Hex format, upper case 'n' - Number format '%' - Percentage format
--	--

Example

x = format(0.5, '%')	Output - 50.000000%
x = format(255, 'x')	Output - ff

User Defined Function

Function Definition

- A function definition is the actual function and start with the “**def**” keyword.
- The definition contains the code that will be executed.
- The first line of a function definition, called the function header, and specifies the parameter list.
- A function header end with a colon.
- The header is the function body, containing the statements that the function will perform.
- The function body consists of indented statements.
- A return statement should be included to specify the returning a value.
- **Example**

```
def hello( ) :
    print("Hello How Are You ! ")
```

Parameters

- The parameter list consists of none or more parameters.
- Parameters are called arguments, if the function is called.
- Parameter can be mandatory or optional.
- The optional parameters (zero or more) must follow the mandatory parameters.
- **Example**

```
def AddOne( x ) :
    return x+1
```

```
n=5  
print( AddOne(n) )      ---- Output is 6
```

Example-1

Program : Write a function to calculate the sum of 2 no and print the result.

```
def sum(a,b):  
    "function to calculate the sum of 2 no"  
    r=a+b  
    print("sum =",r)  
  
a=10  
b=20  
r=sum(a,b)
```

Example-2

Program : Write a function to calculate the sum of 2 no and return the result.

```
def sum(a,b):  
    "function to calculate the sum of 2 no"  
    r=a+b  
    return(r)  
  
a=10  
b=20  
r=sum(a,b)  
print("sum =",r)
```

Example-3

Program : Write a function to find the maximum of 2 no and return the result.

```
def MAX(a,b):  
    "function to calculate the sum of 2 no"  
    if a>b :  
        r=a  
    else:  
        r=b  
    return(r)
```



```
a=10
b=20
r=MAX(a,b)
print("Maximum =",r)
```

Example-4

Note-> Maintaining reference of the passed object and appending values in the same object

Program : Write a function to modify the items in the existing list.

```
def changeme( mylist ):
    "This changes a passed list into this function"
    print ("Values inside the function before change: ", mylist)

    mylist[2]=50
    print ("Values inside the function after change: ", mylist)
    return
```

Now you can call changeme function

```
mylist = [10,20,30]
changeme( mylist )
print ("Values outside the function: ", mylist)
```

Output

```
Values inside the function before change: [10, 20, 30]
Values inside the function after change: [10, 20, 50]
Values outside the function: [10, 20, 50]
```

Example-5

Note-> argument is being passed by reference and the reference is being overwritten inside the called function

Program : Write a function to modify the items in the existing list.

```
def changeme( mylist ):
    "This changes a passed list into this function"

    mylist = [1,2,3,4] # This would assi new reference in mylist
    print ("Values inside the function: ", mylist)
```

return

Now you can call changeme function

```
mylist = [10,20,30]
```

```
changeme( mylist )
```

```
print ("Values outside the function: ", mylist)
```

Output

Values inside the function: [1, 2, 3, 4]

Values outside the function: [10, 20, 30]

Day-9

Q.1 Multiple Choice Question

1. Which one is not true for Dictionary ?
 - a. Items are in (key : value) pair.
 - b. Values are immutable.
 - c. Values must be unique.
 - d. Key can be number or string.

2. Which one is not true for Dictionary?
 - a. `get()` allows to access the value.
 - b. `values()` allows to access all the values.
 - c. `keys()` allows to access all the keys
 - d. `pair()` allows to access all the key-value pair..

3. Which one is not true for Dictionary?
 - a. `d1=d2` allows to copy the one dictionary items to other, with reference.
 - b. `d1=d2` allows to copy the one dictionary items to other, without reference.
 - c. `copy()` method allows to copy the one dictionary items to other, without reference.
 - d. `dict()` method allows to copy the one dictionary items to other, without reference.

4. Which one is not true for Dictionary?
 - a. `pop()` remove the item from dictionary at the specified keyname.
 - b. `popitem()` remove the last item from dictionary.
 - c. `clear()` removes all the items and leave dictionary empty.
 - d. `del` removes all the items and leave dictionary empty

5. Which is not valid for function.?
 - a. Function can return a maximum one value.
 - b. Function with only `PASS` statement, do not give error message.
 - c. `Sum%Num` is a valid function name.
 - d. `SUM_NUM` is a valid function name.

Q.2 . Separate the Keys and Values in the List

Input: d= { 'name' : 'Ajay', 'age' : 40, 'class':5 }

Output

Key = ['name', 'age', 'class']

Value = ['Ajay',40,5]

Q.3 . T= { 1:10,2:20,3:30,4:40,5:50,6:60 }

Perform the following :

- 1) Count the Keys
- 2) Count the Values
- 3) Sum of all the Values

Q.4 Join the lists as Key and Value pair in dictionary

Input

Key = ['Mango', 'Banana', 'Apple']

Value = [40, 50,100]

Output

Fruit_Price = { 'Mango' : 40 , 'Banana' : 50 , 'Apple' : 100 }

Q.5 Write a program to perform the following

input

n= [11,12,13,14,15,21,22,23,24,25]

Output

Dictionary

{ 'odd' : 6 , 'even' : 4 }

Q.6 Write a function to calculate the area of rectangle.

Q.7 Write a function to calculate the area of circle.

Q.8 Write a function to calculate the factorial of a number.

Q.9 Write a function to calculate the sum of first 10 natural number.

Q.10 Write a function to find the sum of 4 numbers and return the result.

7.

Function Arguments

Arguments or Parameters

- Information can be passed into functions as arguments.
- Arguments are specified after the function name, inside the parentheses.
- We can add many arguments separating them with a comma.

Function Arguments

A function can be called by using the following types of formal arguments –

- Required arguments
- Keyword arguments(**kwargs**)
- Default arguments
- Variable-length arguments or Arbitrary Arguments(***args**)
- Arbitrary Keyword Arguments (****kwargs**)

Required arguments

- Required arguments are the arguments passed to a function in correct positional order.
- The number of arguments in the function call should match exactly with the function definition.

Function definition is here

```
def printme( name, age ):  
    "This prints a passed string into this function"  
    print (name , age)
```

Now you can call printme function

```
printme("Ajay",30)
```

To call the function **printme()**, it is definitely need to pass one argument, otherwise it gives a syntax error.

Keyword Arguments (kwargs)

- Keyword arguments in a function call, allows identifying the arguments by the parameter name.
- This allows us to skip arguments or place them out of order. The interpreter is able to use the keywords provided to match the values with parameters.

```
# Function definition is here
def printme( name, age ):
    "This prints a passed string into this function"
    print (name , age)
```

```
# Now you can call printme function
printme(age=30, name="Ajay")
```

Default Arguments

- A default argument is an argument that assumes a default value if a value is not provided in the function call for that argument.

```
# Function definition is here
def printme( name, age=40 ):
    "This prints a passed string into this function"
    print (name , age)
```

```
# Now you can call printme function
printme(age=30, name="Ajay")
printme(name="Ajay")
```

Output

```
Ajay 30
Ajay 40
```

Variable-length Arguments or Arbitrary Arguments(*args)

- If you do not know how many arguments that will be passed into your function, add an asterisk (*) before the parameter name in the function definition.
- This variable name holds the values of all non keyword variable arguments.
- The function will receive a *tuple* of arguments, and can access the items accordingly. This tuple remains empty if no additional arguments are specified during the function call.

Example -1

```
def my_function(*kids):  
    print("The youngest child is " + kids[2])  
  
my_function("Emil", "Tobias", "Linus")
```

Output- The youngest child is Linus

Example -2

```
def printno( arg1, *vartuple ):  
    "This prints a variable passed arguments"  
    print ("Output is: ")  
    print (arg1)  
  
    for var in vartuple:  
        print (var)  
  
printno( 10 )  
printno( 70, 60, 50 )
```

Output is:

10

Output is:

70

60

50

Arbitrary Keyword Arguments (**kwargs)

- If you do not know how many keyword arguments that will be passed into your function, add two asterisk: ** before the parameter name in the function definition.
- This way the function will receive a *dictionary* of arguments, and can access the items accordingly

```
def my_function(**kid):  
    print("His last name is " + kid["lname"])  
  
my_function(fname = "Ajay", lname = "Kumar")
```

Recursion

Python also accepts function recursion, which means a defined function can call itself.

```
def fact(k):  
    "This function returns the factorial of a number"  
  
    if(k > 0):  
        f = k * fact(k - 1)  
    else:  
        f = 1  
    return f  
  
print("\n\nRecursion Example Results")  
r=fact(5)  
print("factorial=",r)
```

The Anonymous Functions

- These functions are called anonymous because they are not declared in the standard manner by using the **def** keyword.
- You can use the **lambda** keyword to create small anonymous functions.
- Lambda forms can take any number of arguments but return just one value in the form of an expression. They cannot contain commands or multiple expressions.
- An anonymous function cannot be a direct call to print because lambda requires an expression.
- Lambda functions have their own local namespace and cannot access variables other than those in their parameter list and those in the global namespace.

Syntax

```
lambda [arg1 [,arg2,.....argn]]:expression
```


Example-1

```
# Function definition is here
sum = lambda arg1, arg2: arg1 + arg2

# Now you can call sum as a function
print ("Value of total : ", sum( 10, 20 ))
print ("Value of total : ", sum( 20, 20 ))
```

Example-2

```
# Function definition is here
def printme(name,age=40):
    "This prints a passed string into this function"
    print (name , age)

# Now you can call printme function
show=lambda x,y:printme(x,y)

printme(age=30, name="Ajay")
show("Ajay",40)
```

Day 10

Q.1 Multiple Choice

1. Which is invalid Function Argument Types?
 - a. Required Argument.
 - b. Keyword Argument.
 - c. Partial Argument.
 - d. Default Argument.
2. Which is true for Arbitrary Argument?
 - a. Allows only numeric values.
 - b. Atleast passing two values are compulsory.
 - c. Allows immutable values only as argument.
 - d. Allows mutable values only as argument.
3. Which is true for Arbitrary Keyword Argument?
 - a. Allows only numeric values.
 - b. Atleast passing two values are compulsory.
 - c. Allows immutable values only as argument.
 - d. Allows any type of value as argument.
4. Which one is not true for Argument?
 - a. Arbitrary argument receives argument as tuple.
 - b. Arbitrary Keyword argument receives argument as dictionary.
 - c. Default argument allows default argument to follows non-default.
 - d. Default argument allows non-default argument to follows default.
5. Which one is not true for Lambda Function?
 - a. It is used to create small function.
 - b. It cannot access global namespace variables.
 - c. It can return only one value.
 - d. It can access global namespace variable in expression.

Q.2 Write a lambda function to find the sum of 4 numbers .

Q.3 Write a lambda function to calculate the area of circle.

Q.4 Write a lambda function to calculate the factorial of a number.

Q.5 Write a function to calculate the sum of first 10 natural number.

Q.6 Write a lambda function to find the maximum from items of List collection passed as argument.

Q.7 Write a function to find the maximum from the numbers passed as argument to the function. [use Arbitrary Argument]

Q.8 Write a function to find the person eligible to vote. Details of the person (name , age , address) passed as argument. [use the Arbitrary Keyword Argument]

Q.9 Write a function to find the sum of all the numbers only passed as argument. The function can accept the arguments of any type. [Use Arbitrary Argument]

Example SUM(10, 20, 'Ajay', 30, '#rr', 40, '50')

In this case, only 10, 20, 30, 40 is numeric, so the function will return the 100.

Q.10 Write a function to find the count of numbers in the range specified below for the 10 numbers passed as argument and return result.

Count for < 0
Count for 0 - 50
Count for 51 – 100
Count for > 100

Example

-78	89	76	12	45	67	78	91	2	-5
-----	----	----	----	----	----	----	----	---	----

Count for < 0 : 2
Count for 0 - 50 : 3
Count for 51 – 100 : 5
Count for > 100 : 0

Contents to be covered

1. Mean , Median , Mode
2. Standard Deviation
3. Variance

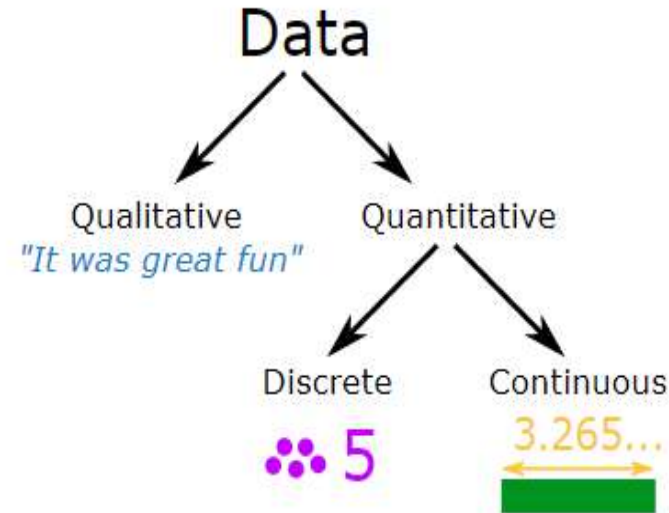
Data

Data is a collection of facts, such as numbers, words, measurements, observations or just descriptions of things.

Qualitative vs Quantitative

Data can be qualitative or quantitative.

- **Qualitative data** is descriptive information (it *describes* something)
- **Quantitative data** is numerical information (numbers)



Data

Quantitative data can be Discrete or Continuous:

- **Discrete data** can only take certain values (like whole numbers)
- **Continuous data** can take any value (within a range)

Put simply: **Discrete data** is counted, **Continuous data** is measured

Example

Qualitative:

- Your friends' favorite holiday destination
- The most common given names in your town
- How people describe the smell of a new perfume

Quantitative:

- Height (Continuous)
- Weight (Continuous)
- Petals on a flower (Discrete)
- Customers in a shop (Discrete)

Qualitative:

- He is brown and black
- He has long hair
- He has lots of energy

Quantitative:

- Discrete:
 - He has 4 legs
 - He has 2 brothers
- Continuous:
 - He weighs 25.5 kg
 - He is 565 mm tall

Data Collection

Collecting

Data can be collected in many ways. The simplest way is direct observation.

Example: Counting Cars

You want to find how many cars pass by a certain point on a road in a 10-minute interval.

Census or Sample

A **Census** is when we collect data for **every** member of the group (the whole "population").

A **Sample** is when we collect data just for **selected members** of the group.

Example: 120 people in your local football club

You can ask everyone (all 120) what their age is. That is a census.

Or you could just choose the people that are there this afternoon. That is a sample.

A census is accurate, but hard to do. A sample is not as accurate, but may be good enough, and is a lot easier.

Measures of Central Value

Finding a Central Value

Mean Value

Median Value

Mode or Modal Value

Central Value

Finding a Central Value

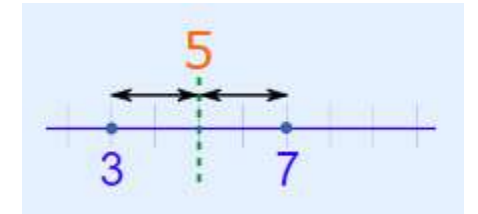
When you have two or more numbers it is nice to find a value for the "center".

2 Numbers

With just 2 numbers the answer is easy: go half-way between.

Example: what is the central value for 3 and 7?

Answer: Half-way between, which is 5.



We can calculate it by adding 3 and 7 and then dividing the result by 2:

$$(3+7) / 2 = 10/2 = 5$$

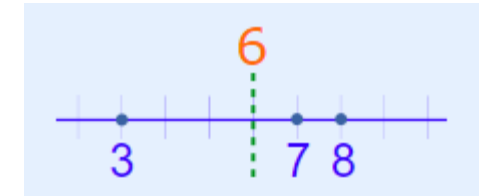
3 or More Numbers

We can use that idea of "adding then dividing" when we have 3 or more numbers:

Example: what is the central value of 3, 7 and 8?

We calculate it by adding 3, 7 and 8 and then dividing the results by 3 :

$$(3+7+8) / 3 = 18/3 = 6$$



Mean

Mean: Add up the numbers and divide by how many numbers.

Example: Birthday Activities

Uncle Bob wants to know the average age at the party, to choose an activity.

There will be 6 kids aged 13, and also 5 babies aged 1.
Add up all the ages, and divide by 11 (because there are 11 numbers):

$$(13+13+13+13+13+13+1+1+1+1+1) / 11 = 7.5...$$



The mean age is about $7\frac{1}{2}$, so he gets a **Jumping Castle!**

The 13 year olds are embarrassed,
and the 1 year olds can't jump!

The Median

simply list all numbers in order and choose the middle one

Example: Birthday Activities (continued)

List the ages in order:

1, 1, 1, 1, 1, 13, 13, 13, 13, 13, 13

Choose the middle number:

1, 1, 1, 1, 1, **13**, 13, 13, 13, 13, 13

The Median age is **13** ... so let's have a **Disco!**

Sometimes there are **two** middle numbers. Just average those two:

Example: What is the Median of 3, 4, 7, 9, 12, 15

There are two numbers in the middle:

3, 4, 7, 9, 12, 15

So we average them:

$$(7+9) / 2 = 16/2 = 8$$

The Median is **8**

Mode

The Mode is the value that occurs most often.

Example: Birthday Activities (continued)

Group the numbers so we can count them:

1, 1, 1, 1, 13, 13, 13, 13, 13, 13

"13" occurs 6 times, "1" occurs only 5 times, so the mode is **13**.

But Mode can be tricky, there can sometimes be more than one Mode.

Example: What is the Mode of 3, 4, 4, 5, 6, 6, 7

Well ... 4 occurs twice but 6 **also** occurs twice.

So **both 4 and 6** are modes.

When there are two modes it is called "bimodal", when there are three or more modes we call it "multimodal".

Outliers

Outliers are values that "**lie outside**" the other values.

They can change the mean a lot, so we can either not use them (and say so) or use the median or mode instead.

Example: 3, 4, 4, 5 and 104

Mean: Add them up, and divide by 5 (as there are 5 numbers):

$$(3+4+4+5+104) / 5 = \mathbf{24}$$

24 does not represent those numbers well at all!

Without the 104 the mean is:

$$(3+4+4+5) / 4 = \mathbf{4}$$

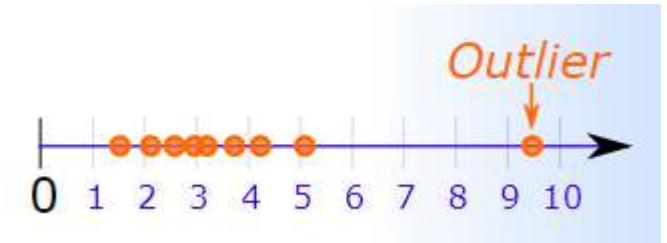
But please tell people you are not including the outlier.

Median: They are in order, so just choose the middle number, which is **4**:

3, 4, **4**, 5, 104

Mode: 4 occurs most often, so the Mode is **4**

3, **4**, **4**, 5, 104



Grouping

In some cases (such as when all values appear the same number of times) the mode is not useful. But we can **group** the values to see if one group has more than the others.

Example: {4, 7, 11, 16, 20, 22, 25, 26, 33}

Each value occurs once, so let us try to group them.

We can try groups of 10:

0-9: **2 values** (4 and 7)

10-19: **2 values** (11 and 16)

20-29: **4 values** (20, 22, 25 and 26)

30-39: **1 value** (33)

In groups of 10, the "20s" appear most often, so we could choose **25** (the middle of the 20s group) as the mode. You could use different groupings and get a different answer.

Grouping also helps to find what the typical values are when the real world messes things .

Grouping

Example: How long to fill a pallet?

Its recorded , how long it takes to fill a pallet (in minutes) :
{35, 36, 32, 42, 58, 56, 35, 39, 46, 47, 34, 37}

It takes longer when there is break time or lunch so an average is not very useful.

But grouping by 5s gives:

30-34: **2**

35-39: **5**

40-44: **1**

45-49: **2**

50-54: **0**

54-59: **2**

"35-39" appear most often, so we can say it normally takes **about 37 minutes** to fill a pallet.

Mean, Median and Mode

Mr. Ajay timed 21 people in the sprint race, to the nearest second:

59, 65, 61, 62, 53, 55, 60, 70, 64, 56, 58, 58, 62, 62, 68, 65, 56, 59, 68, 61, 67

To find the **Mean** Alex adds up all the numbers, then divides by how many numbers:

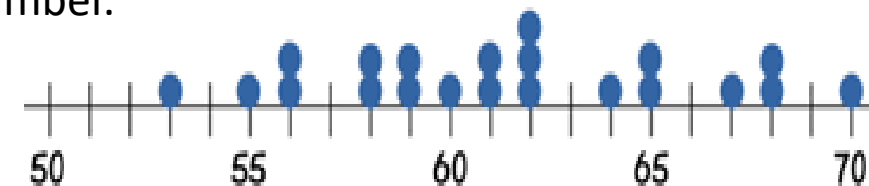
$$\begin{aligned} \text{Mean} &= (59 + 65 + 61 + 62 + 53 + 55 + 60 + 70 + 64 + 56 + 58 + 58 + 62 + 62 + 68 + 65 + 56 + 59 + 68 + 61 + 67) / 21 \\ &= 61.38095... \end{aligned}$$

To find the **Median**, places the numbers in value order and finds the middle number.

In this case the median is the 11th number:

53, 55, 56, 56, 58, 58, 59, 59, 60, 61, 61, 62, 62, 62, 64, 65, 65, 67, 68, 68, 70

Median = 61



To find the **Mode**, or modal value, places the numbers in value order then counts how many of each number. The Mode is the number which appears most often (there can be more than one mode):

53, 55, 56, 56, 58, 58, 59, 59, 60, 61, 61, 62, 62, 62, 64, 65, 65, 67, 68, 68, 70

62 appears three times, more often than the other values, so **Mode = 62**

Range , Quartiles

The **Range** is the difference between the lowest and highest values.

Example: In {4, 6, 9, 3, 7}

the lowest value is 3, and
the highest is 9.
So the range is $9 - 3 = 6$.

Example: In {8, 11, 5, 9, 7, 6, 3616} :

the lowest value is 5, and
the highest is 3616,

So the range is $3616 - 5 = 3611$.

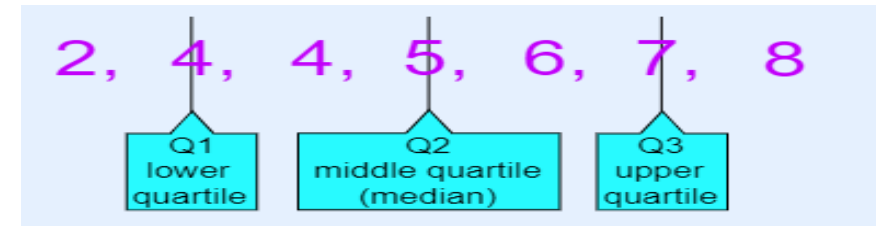
Quartiles are the values that divide a list of numbers into quarters:

- Put the list of numbers **in order**
- Then cut the list into **four equal parts**
- The Quartiles are at the "cuts"

Example: 5, 7, 4, 4, 6, 2, 8

Put them in order: 2, 4, 4, 5, 6, 7, 8

Cut the list into quarters:



And the result is:

- Quartile 1 (Q1) = 4
- Quartile 2 (Q2), which is also the Median, = 5
- Quartile 3 (Q3) = 7

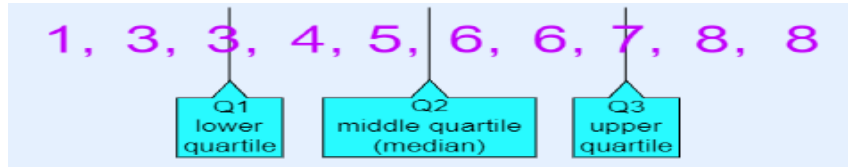
Range , Quartiles, Interquartile Range

Quartiles

Example: 1, 3, 3, 4, 5, 6, 6, 7, 8, 8

The numbers are already in order

Cut the list into quarters:



In this case Quartile 2 is half way between 5 and 6:

$$Q2 = (5+6)/2 = 5.5$$

And the result is:

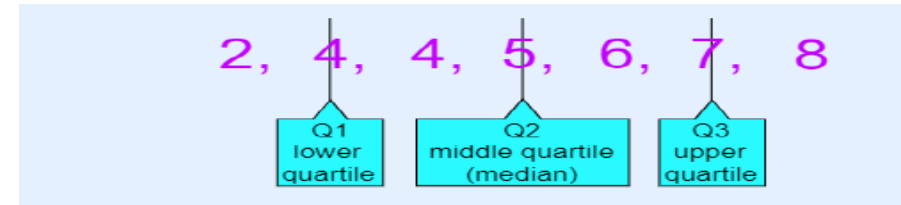
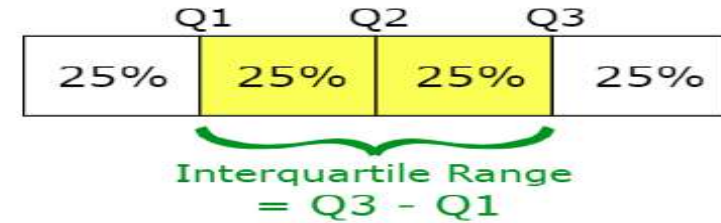
$$\text{Quartile 1 (Q1)} = 3$$

$$\text{Quartile 2 (Q2)} = 5.5$$

$$\text{Quartile 3 (Q3)} = 7$$

Interquartile Range

The "Interquartile Range" is from Q1 to Q3:

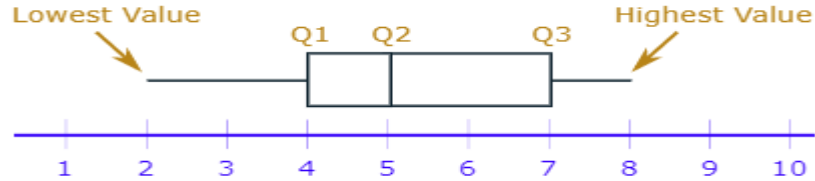


The **Interquartile Range** is:

$$Q3 - Q1 = 7 - 4 = 3$$

Box and Whisker Plot

Important values in a "Box and Whisker Plot":



Example: **Box and Whisker Plot and Interquartile Range** for

4, 17, 7, 14, 18, 12, 3, 16, 10, 4, 4, 11

Put them in order:

3, 4, 4, 4, 7, 10, 11, 12, 14, 16, 17, 18

Cut it into quarters:

3, 4, 4 | 4, 7, 10 | 11, 12, 14 | 16, 17, 18

In this case all the quartiles are between numbers:

Quartile 1 (Q1) = $(4+4)/2 = 4$

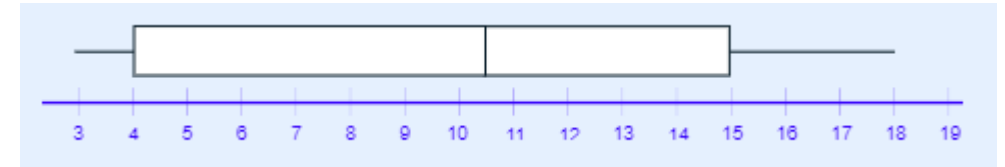
Quartile 2 (Q2) = $(10+11)/2 = 10.5$

Quartile 3 (Q3) = $(14+16)/2 = 15$

Also:

The Lowest Value is **3**, The Highest Value is **18**

Box and Whisker Plot:



And the **Interquartile Range** is:

$$Q3 - Q1 = 15 - 4 = 11$$

Mean Deviation

How far, on average, all values are from the middle.

In three steps:

1. Find the mean of all values
2. Find the **distance** of each value from that mean (subtract the mean from each value, ignore minus signs)
3. Then find the **mean of those distances**

Mean Deviation

Example: The Mean Deviation of 3, 6, 6, 7, 8, 11, 15, 16

Step 1: Find the **mean**:

$$\text{Mean} = (3 + 6 + 6 + 7 + 8 + 11 + 15 + 16) / 8 = 72 / 8 = 9$$

Step 2: Find the **distance** of each value from that mean:

Which looks like this:

Step 3. Find the **mean of those distances**:

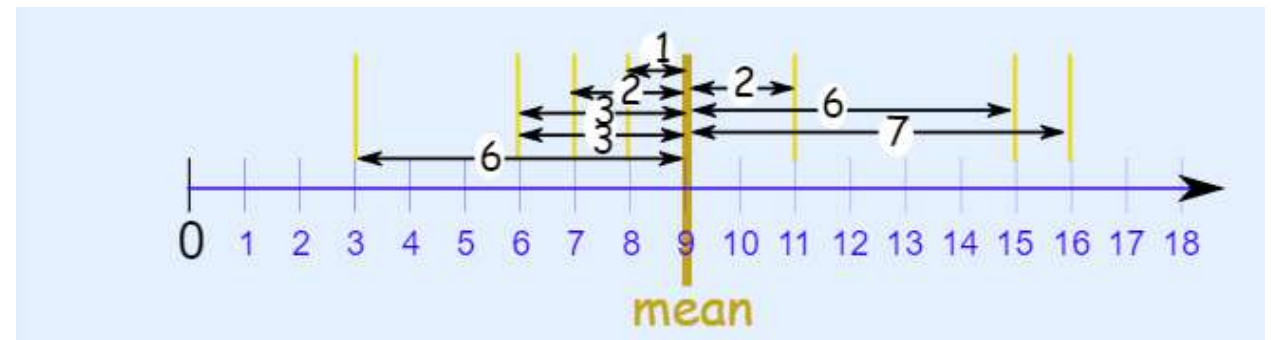
$$\text{Mean Deviation} = (6 + 3 + 3 + 2 + 1 + 2 + 6 + 7) / 8 = 30 / 8 = 3.75$$

So, the **mean = 9**, and the **mean deviation = 3.75**

It tells us how far, on average, all values are from the middle.

In that example the values are, on average, 3.75 away from the middle.

Value	Distance from 9
3	6
6	3
6	3
7	2
8	1
11	2
15	6
16	7



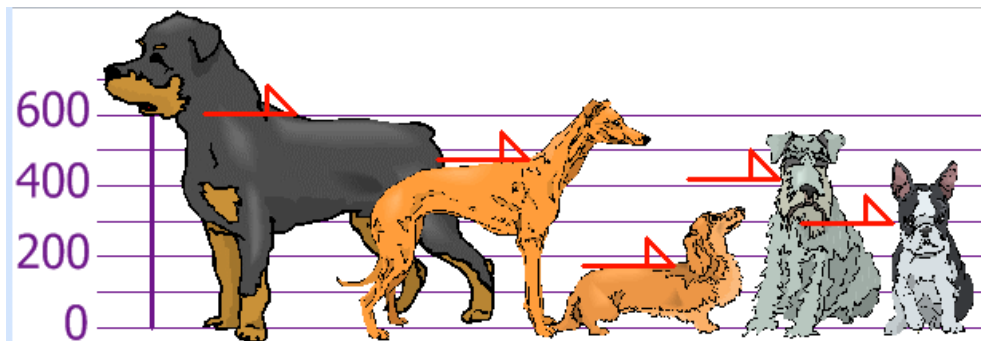
Mean Deviation

Formula

The formula is:

$$\text{Mean Deviation} = (\Sigma |x - \mu|) / N$$

Σ is Sigma, which means to sum up
 || (the vertical bars) mean Absolute Value,
 basically to ignore minus signs
 x is each value (such as 3 or 16)
 μ is the mean (in our example $\mu = 9$)
 N is the number of values ($N = 8$)



Example –

The heights (at the shoulders) are: 600mm, 470mm, 170mm, 430mm and 300mm.

Step 1: Find the **mean**:

$$\mu = 600 + 470 + 170 + 430 + 300 / 5 = 1970 / 5 = 394$$

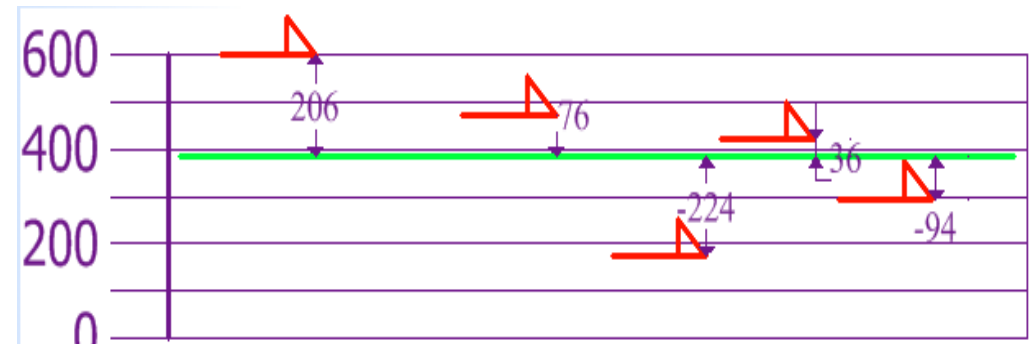
Step 2: Find the **Absolute Deviations**:

Step 3. Find the **Mean Deviation**:

$$\text{Mean Deviation} = \Sigma |x - \mu| / N = 636 / 5 = 127.2$$

x	$ x - \mu $
600	206
470	76
170	224
430	36
300	94
$\Sigma x - \mu = 636$	

So, on average, the dogs' heights are **127.2 mm** from the mean.



Standard Deviation

Standard Deviation

The Standard Deviation is a measure of how spread out numbers are.

Its symbol is σ (the greek letter sigma)

The formula : it is the **square root** of the **Variance**. So now you ask, "What is the Variance?"

Variance

The Variance is defined as:

The average of the **squared** differences from the Mean.

To calculate the variance follow these steps:

- Work out the Mean (the simple average of the numbers)
- Then for each number: subtract the Mean and square the result (the *squared difference*).
- Then work out the average of those squared differences.

Formulas

Here are the two formulas, explained at [Standard Deviation Formulas](#) if you want to know more:

The "Population Standard Deviation":
$$\sigma = \sqrt{\frac{1}{N} \sum_{i=1}^N (x_i - \mu)^2}$$

The "Sample Standard Deviation":
$$s = \sqrt{\frac{1}{N-1} \sum_{i=1}^N (x_i - \bar{x})^2}$$

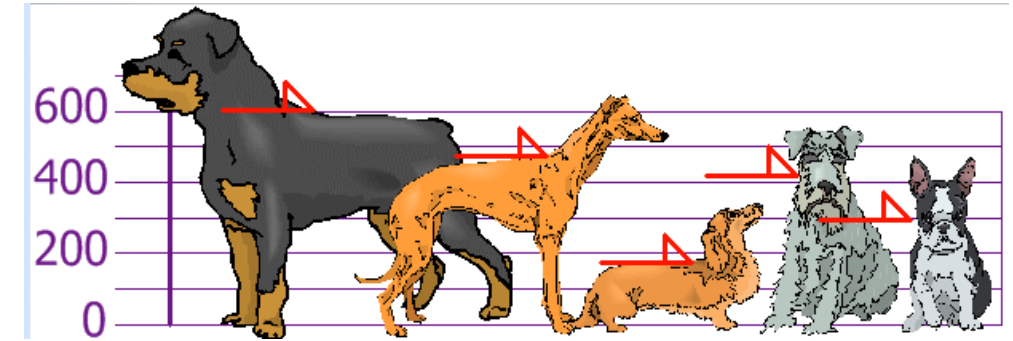
Looks complicated, but the important change is to divide by **N-1** (instead of **N**) when calculating a Sample Variance.

Standard Deviation

Example

You and your friends have just measured the heights of your dogs (in millimeters):

The heights (at the shoulders) are: 600mm, 470mm, 170mm, 430mm and 300mm.



Find out the Mean, the Variance, and the Standard Deviation.

Your first step is to find the Mean:

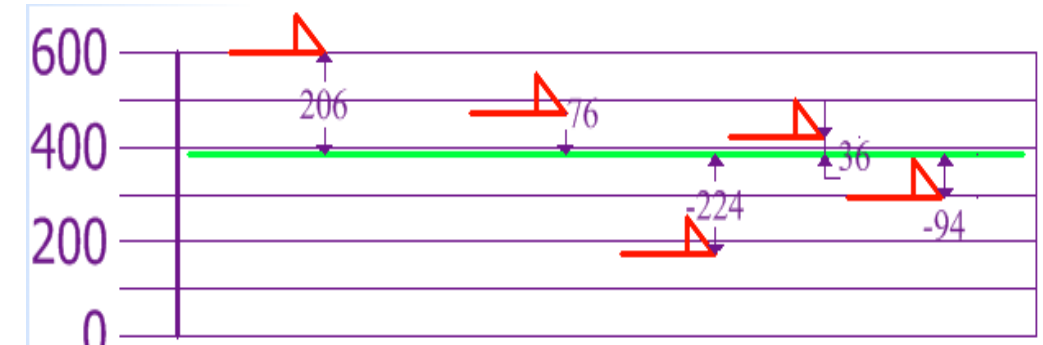
Answer:

$$\text{Mean} = \frac{600 + 470 + 170 + 430 + 300}{5}$$

$$= \frac{1970}{5} = 394$$

so the mean (average) height is 394 mm. Let's plot this on the chart:

Now we calculate each dog's difference from the Mean:



Standard Deviation

To calculate the Variance, take each difference, square it, and then average the result:

Variance

$$\begin{aligned}\sigma^2 &= \frac{206^2 + 76^2 + (-224)^2 + 36^2 + (-94)^2}{5} \\ &= \frac{42436 + 5776 + 50176 + 1296 + 8836}{5} \\ &= \frac{108520}{5} \\ &= 21704\end{aligned}$$

So the Variance is **21,704**

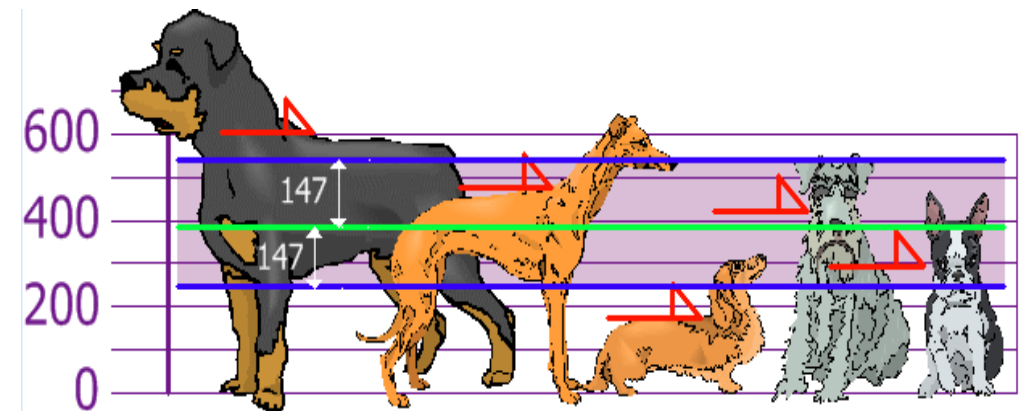
And the Standard Deviation is just the square root of Variance, so:

Standard Deviation

$$\begin{aligned}\sigma &= \sqrt{21704} \\ &= 147.32... \\ &= \mathbf{147} \text{ (to the nearest mm)}\end{aligned}$$

The Standard Deviation is useful.

Now we can show which heights are within one Standard Deviation (147mm) of the Mean.



So, using the Standard Deviation we have a "standard" way of knowing what is normal, and what is extra large or extra small.

Rottweilers **are** tall dogs. And Dachshunds **are** a bit short, right?

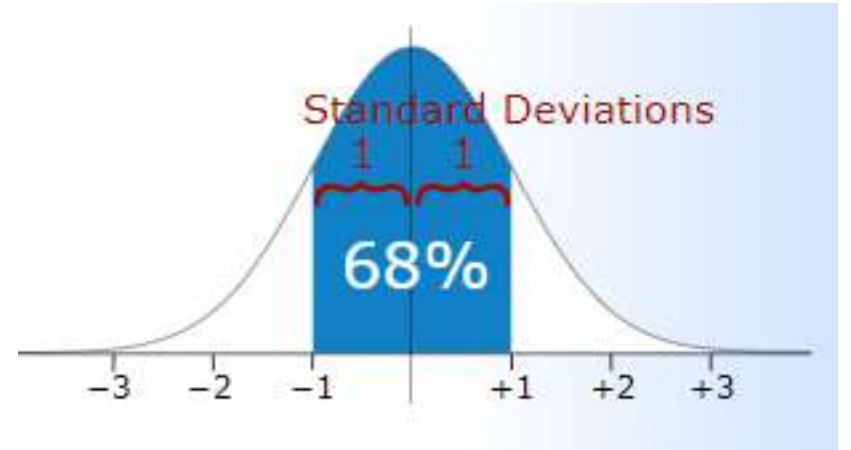
Standard Deviation

We can expect about 68% of values to be within plus-or-minus 1 standard deviation.

But if the data is a **Sample** (a selection taken from a bigger Population), then the calculation changes!

When you have "N" data values :

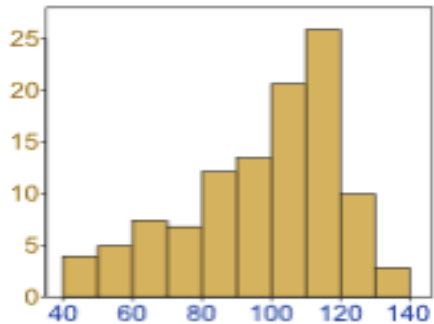
- **The Population:** divide by **N** when calculating Variance
- **A Sample:** divide by **N-1** when calculating Variance



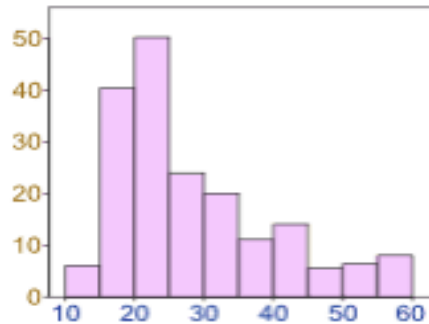
Normal Distribution

Data can be "distributed" (spread out) in different ways.

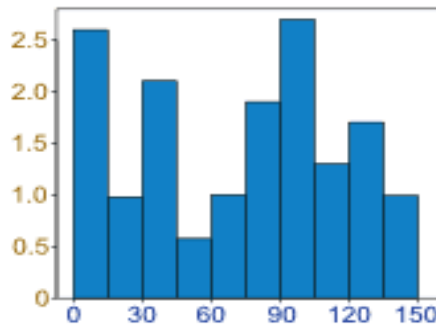
It can be spread out
more on the left



Or more on the right

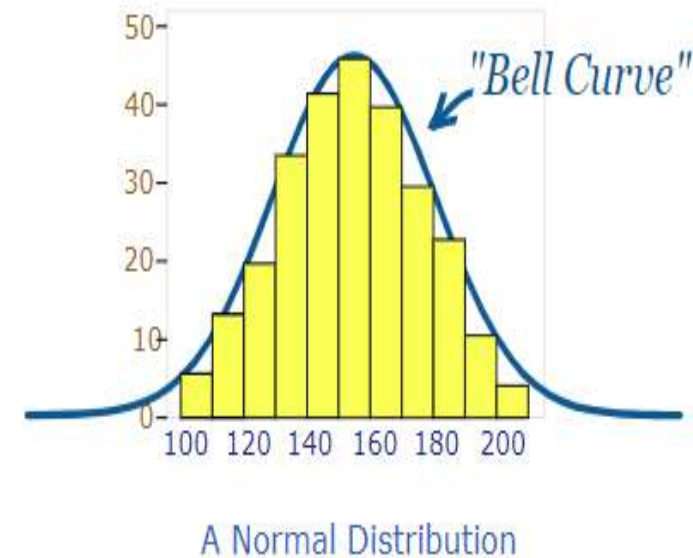


Or it can be all jumbled up



But there are many cases where the data tends to be around a central value with no bias left or right, and it gets close to a "Normal Distribution" like this:

The "Bell Curve" is a Normal Distribution.



Normal Distribution

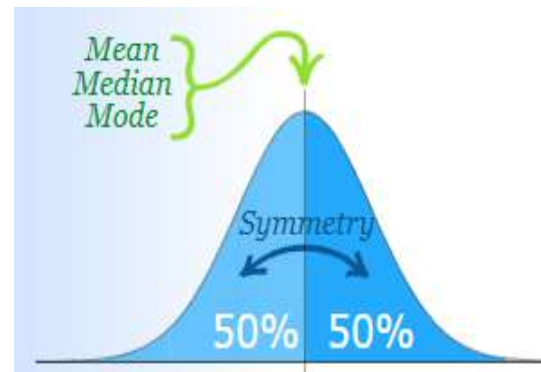
Many things closely follow a Normal Distribution:

- heights of people
- size of things produced by machines
- errors in measurements
- blood pressure
- marks on a test

We say the data is "normally distributed":

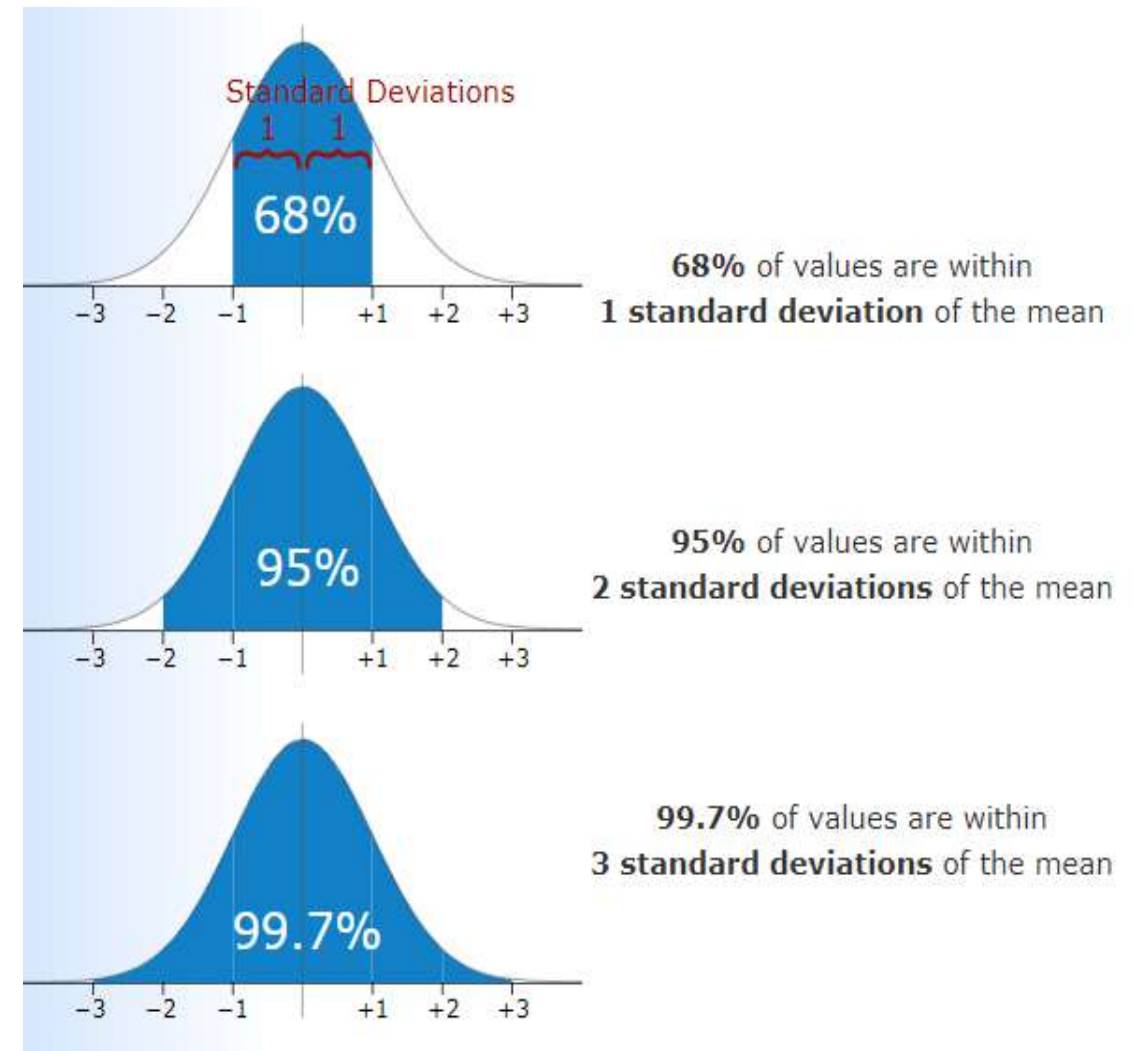
The **Normal Distribution** has:

- mean = median = mode
- symmetry about the center
- 50% of values less than the mean and 50% greater than the mean



Standard Scores

The number of **standard deviations from the mean** is also called the "**Standard Score**", "**sigma**" or "**z-score**".



Example -1

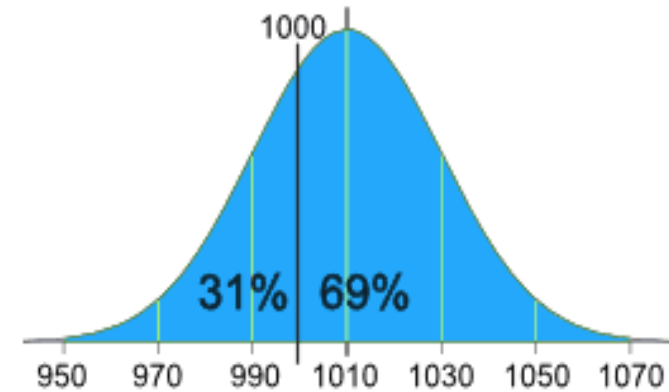
- It can help us make decisions about our data.
- Example: Professor is marking a test.
- Here are the students' results (out of 60 points):
- 20, 15, 26, 32, 18, 28, 35, 14, 26, 22, 17
- Most students didn't even get 30 out of 60, and **most will fail**.
- The test must have been really hard, so the Prof decides to Standardize all the scores and only fail people more than 1 standard deviation below the mean.
- The **Mean is 23**, and the **Standard Deviation is 6.6**, and these are the Standard Scores:
- -0.45, -1.21, 0.45, 1.36, -0.76, 0.76, 1.82, -1.36, 0.45, -0.15, -0.91
- Now only 2 students will fail (the ones lower than -1 standard deviation)
- **Much fairer!**

Example -2

- Your company packages sugar in 1 kg bags.
- When you weigh a sample of bags you get these results:
- 1007g, 1032g, 1002g, 983g, 1004g, ... (a hundred measurements)
- Mean = 1010g , Standard Deviation = 20g

Some values are less than 1000g ... can you fix that?

- The normal distribution of your measurements looks like this:
- It is a random thing, so we can't **stop** bags having less than 1000g, but we can try to **reduce it** a lot.



31% of the bags are less than 1000g,
which is cheating the customer!

Let's adjust the machine so that 1000g is:

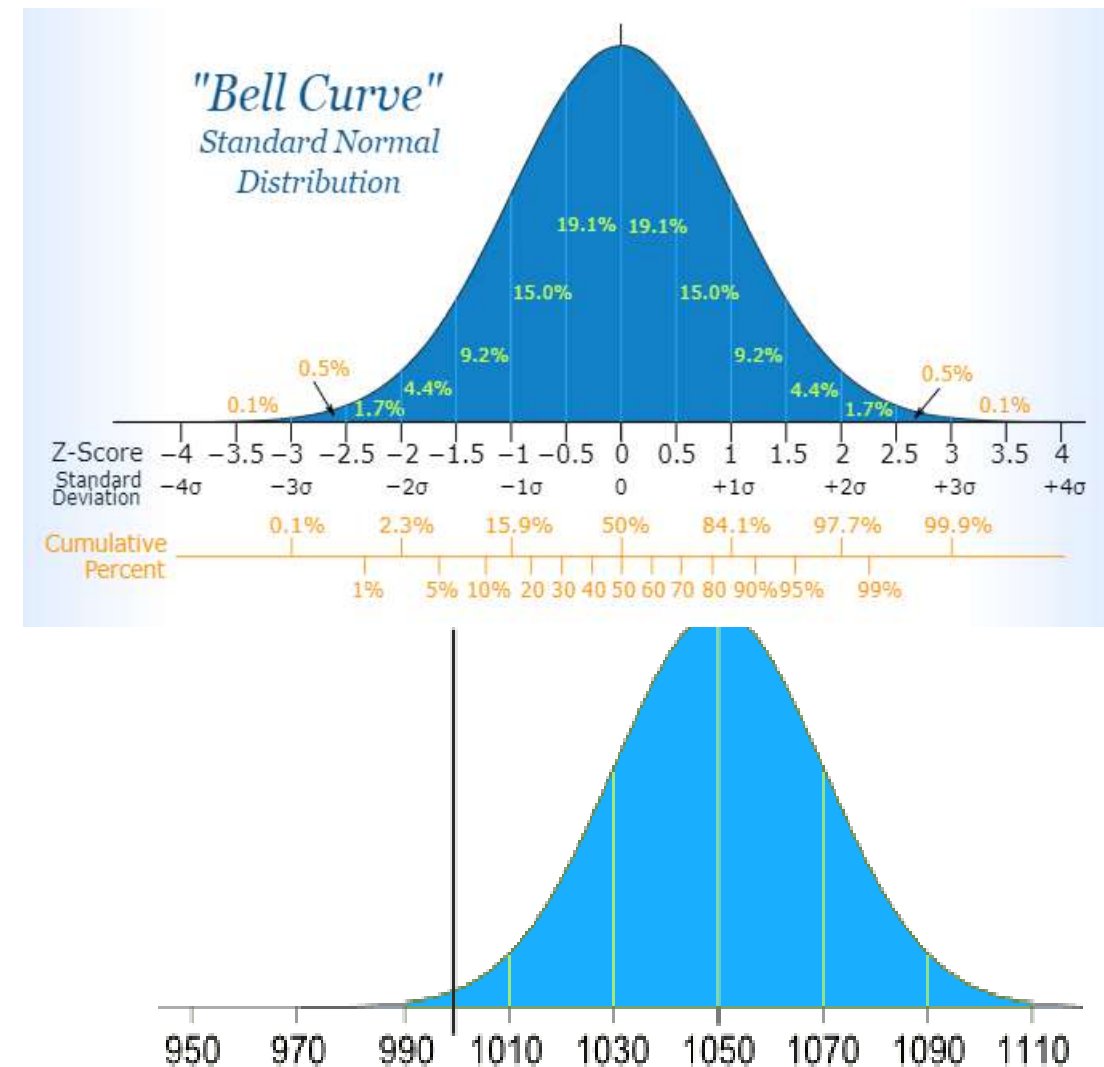
- at -3 standard deviations:
From the big bell curve above we see that **0.1%** are less. But maybe that is too small.
- at -2.5 standard deviations:
Below 3 is 0.1% and between 3 and 2.5 standard deviations is 0.5%, together that is $0.1\% + 0.5\% = \mathbf{0.6\%}$

So let us adjust the machine to have **1000g at -2.5 standard deviations** from the mean.

The standard deviation is 20g, and we need 2.5 of them:

$$2.5 \times 20\text{g} = 50\text{g}$$

So the machine should average 1050g, like this:



Day -11

Q.1 Multiple Choice

1. Which one is not true for Scope of Object?
 - a. In-Scope – Object which are accessible.
 - b. Out-Scope – Object which are not accessible.
 - c. Local Scope- Variable assigned out of one function.
 - d. Global Scope – Variable assigned value out of all the function, at top level.

2. Which one is not true for Namespace?
 - a. Built-in Namespace.
 - b. Global Namespace.
 - c. Local Namespace.
 - d. Function Namespace.

3. Which one is not true for LEGB Rule?
 - a. Global Scope is the top most scope.
 - b. Built-in Scope contains the keywords that are built into Python.
 - c. Local scope contains the names defined outside the function.
 - d. Enclosing Scope contains the names defined in nested function.

4. Which one is not true for Module ?
 - a. **import** statement used to include the object from Module.
 - b. **from..import..** statement used to include the specific object from Module.
 - c. **from..import *** statement used to include the specific object from Module.
 - d. Dot notation is used to access the object in import <Module> only.

5. Which one is not true for Package?
 - a. **from..import..** statement used to include the specific module from package.
 - b. **import *** statement allows including all the object from Package
 - c. Package can have multiple modules.
 - d. Package have **`__init__.py`** for initialization of module in Packages.

Q.2 Create a module – My_Number with the following function in it

- a) Sum of 2 number
- b) Average of 3 Number
- c) Factorial of a Number.

Using of NumPy

Example

```
import numpy as np
arr1=np.array(1)
arr2=np.array([1,2,3,4,5])
print('Numpy Version : ', np.__version__)
print('Array 1 Values : ', arr1)
print('Array 2 Values : ', arr2)
print('Array 1 Data Type : ', type(arr1))
print('Array 2 Data Type : ',type(arr2))
```

Output

```
Numpy Version : 1.18.4
Array 1 Values : 1
Array 2 Values : [1 2 3 4 5]
Array 1 Data Type : <class 'numpy.ndarray'>
Array 2 Data Type : <class 'numpy.ndarray'>
```

Dimension in NumPy

- A dimension in arrays is one level of array depth.
- Types of Arrays

0-D arrays : It has a scalar element in an array. Each value in an array is a 0-D array.

e.g. `numpy.array(10)`

1-D arrays : An array that has 0-D arrays as its elements is called uni-dimensional or 1-D array.

e.g. `numpy.array([30 , 40])`

2-D arrays : An array that has 1-D arrays as its elements is called a 2-D array.

e.g. `numpy.array([[30 , 40] , [50 , 60]])`

3-D arrays : An array that has 2-D arrays (matrices) as its elements is called 3-D array.

e.g. `numpy.array([[[30 , 40] , [50 , 60]] ,
[[70 , 80] , [90 , 10]]
])`

- `<array_name>.ndim` used to know the dimension of the array.

Example-1

1D Array

```
import numpy as np
arr1=np.array(1)
arr2=np.array([1,2,3,4,5])
print('Array -1')
print(arr1)
print('Array Dim :', arr1.ndim)
print('Array -2')
print(arr2)
print('Array Dim :', arr2.ndim)
```

Output

```
Array -1
1
```

Array Dim : 0

Array -2

[1 2 3 4 5]

Array Dim : 1

Example

2D Array

import numpy

```
arr3=numpy.array( [ [1,2,3], [4,5,6] ] )
```

```
print('Array -3')
```

```
print(arr3)
```

```
print('Array Dim :', arr3.ndim)
```

Output

Array -3

```
[[1 2 3]
```

```
 [4 5 6]]
```

Array Dim : 2

Example

3D Array

import numpy

```
arr4=numpy.array(
```

```
    [ [ [1,2,3], [4,5,6] ],
```

```
      [ [1,2,3], [4,5,6] ]
```

```
    ] )
```

```
print('Array -4')
```

```
print(arr4)
```

```
print('Array Dim :', arr4.ndim)
```

Output

Array -4

```
[ [ [1 2 3]
```

```
     [4 5 6] ]
```

```
    [ [1 2 3]
```

```
       [4 5 6] ] ]
```

Array Dim : 3

Access Array in NumPy

Slicing for Accessing Array Elements

We can access the elements of array by using the slicing [start: end] or [start : end : step].

[1 : 4] means starting from 1 to 3 (4 is not included).

[: 4] means starting from 0 to 3.

[1 :] means starting from 1 upto end.

[1 : 4 : 1] means 1 , 2 , 3 . step 1 indicates the change in the index value

[1 : 4 : 2] means 1, 3 . step value is 2.
[3 : 0 : -1] means 3, 2,1 . step value is -1.
[-1 : -3 : -1] means -1, -2. step value is -1.

Example -1

```
# Slicing on 1D Array
import numpy as np
a = np.array([ 1, 2, 3, 4 ])
print('Array Elements')
print(a)
print('Dimension of Array: ', a.ndim)
print('Array Elements using Slicing Index')
print('L-1 :', a[1:4])
print('L-2 :', a[:4])
print('L-3 :', a[1: ])
print('L-4 :', a[ : ])
print('L-5 :', a[1:4:1])
print('L-6 :', a[1:4:2])
print('L-7 :', a[3:0:-1])
print('L-8 :', a[-1:-3:-1])
```

Array a	1	2	3	4
Index	0	1	2	3
Negative Index	-4	-3	-2	-1

Output

Array Elements using Slicing Index

L-1 : [2 3 4]
L-2 : [1 2 3 4]
L-3 : [2 3 4]
L-4 : [1 2 3 4]
L-5 : [2 3 4]
L-6 : [2 4]
L-7 : [4 3 2]
L-8 : [4 3]

Example -2

```
#Slicing on 2D Array
import numpy as np
a = np.array([ [1, 2, 3,4,5] , [6,7,8,9,10] ])
print('Array Elements')
print(a)
print('Dimension of Array: ', a.ndim)
print('Array Elements using Slicing Index')
print('L-1 :', a[1,1:4])
print('L-2 :', a[1, :4])
print('L-3 :', a[1, 1:])
print('L-4 :', a[1, :])
print('L-5 :', a[1,1:4:1])
```

```
print('L-6 :', a[1,1:4:2])
print('L-7 :', a[1,3:0:-1])
print('L-8 :', a[1,-1:-3:-1])
print('L-9 :', a[ 0:2 , 0:2])
```

2-D Array

Index	0	1	2	3	4
0	1	2	3	4	5
1	6	7	8	9	10

Array Elements using Slicing Index

```
L-1 : [7 8 9]
L-2 : [6 7 8 9]
L-3 : [ 7 8 9 10]
L-4 : [ 6 7 8 9 10]
L-5 : [7 8 9]
L-6 : [7 9]
L-7 : [9 8 7]
L-8 : [10 9]
L-9 : [ [1 2]
        [6 7] ]
```

Creating Matrix

1. `empty()` - creates an uninitialized array of specified shape and dtype.

```
import numpy as np
x = np.empty([3,2], dtype = int)
print (x)
```

Output

```
[ [12846240  12859408]
  [12875144  12875480]
  [12875480    0] ]
```

2. `ones()` - New array of specified size and type, filled with ones. Default data type is float.

```
import numpy as np
x = np.ones( [2,2] )
print( x)
```

Output

```
[ [1.  1.]
  [1.  1.] ]
```

3. `zeros()` - New array of specified size, filled with zeros.

```
import numpy as np
x = np.zeros(5)
print(x)
x = np.zeros( [3,3] , int)
print(x)
```

Output

```
[0. 0. 0. 0. 0.]  
[[0 0 0]  
 [0 0 0]  
 [0 0 0]]
```

Axis of Matrix

1D Array

Axis=0 represent the columns

Axis=1 represent the rows

Axis	0		
	1	2	3

2D Array

Axis=0 represent the columns

Axis=1 represent the rows

Axis	0		
1	1	2	3
	4	5	6
	7	8	9

Aggregate Function

Numpy has fast built-in **aggregate and statistical** for working on arrays.

<u>SNo</u>	Function	Description
1	np.sum(Arr)	Find the sum of the array.
2	np.prod(Arr)	Find the product of the values of array.
3	np.mean(Arr)	Find the mean of the array.
4	np.std(Arr)	Find the standard deviation of the array.
5	np.var(Arr)	Find the variance of the data of array.
6	np.min(Arr)	Find the minimum from the array.
7	np.max(Arr)	Find the maximum from the array.
8	np.argmin(Arr)	Find the index of minimum value in the array.
9	np.argmax(Arr)	Find the index of maximum value in the array.
10	np.median(Arr)	Find the median of the values of the array.
11	np.average(Arr)	Find the average of the values of the array.

1. `sum()` – calculate the sum of elements in array.

`import numpy as np`

`A1=np.array([1,2,3,4,5])`

`A2=np.array([[1,2,3], [4,5,6]])`

`A3=np.array([[1,2,3], [4,5,6] , [7,8,9]])`

#1D Array – A1

`print(np.sum(A1))` ---- 15

#2D Array – A2

`print(np.sum(A2))` ---- 21

`print(np.sum(A2,0))` ---- [5 7 9]

`print(np.sum(A2,1))` ---- [6 15]

#2D Array – A3

`print(np.sum(A3))` ---- 45

`print(np.sum(A3,0))` ---- [12 15 18]

`print(np.sum(A3,1))` ---- [6 15 24]

Array – A1

Axis =0				
1	2	3	4	5

Array – A2

Axis	0		
1	1	2	3
	4	5	6

Array – A3

Axis	0		
1	1	2	3
	4	5	6
	7	8	9

2. `prod()` – calculate the product of elements in array.

`import numpy as np`

`A1=np.array([1,2,3,4,5])`

`A2=np.array([[1,2,3], [4,5,6]])`

`A3=np.array([[1,2,3], [4,5,6] , [7,8,9]])`

#1D Array – A1

`print(np.prod(A1))` ---- 120

#2D Array – A2

`print(np.prod(A2))` ---- 720

`print(np.prod(A2,0))` ---- [4 10 18]

`print(np.prod(A2,1))` ---- [6 120]

#2D Array – A3

`print(np.prod(A3))` ---- 362880

`print(np.prod(A3,0))` ---- [28 80 162]

`print(np.prod(A3,1))` ---- [6 120 504]

Array - A1

Axis =0				
1	2	3	4	5

Array - A2

Axis	0		
1	1	2	3
	4	5	6

Array - A3

Axis	0		
1	1	2	3
	4	5	6
	7	8	9

3. . max() – calculate the product of elements in array.

import numpy as np

A1=np.array([1,2,3,4,5])

A2=np.array([[1,2,3], [4,5,6]])

A3=np.array([[1,2,3], [4,5,6] , [7,8,9]])

#1D Array – A1

print(np.max(A1)) ---- 5

#2D Array – A2

print(np.max(A2)) ---- 6

print(np.max(A2, 0)) ---- [4 5 6]

print(np.max(A2, 1)) ---- [3 6]

#2D Array – A3

print(np.max(A3)) ---- 9

print(np.max(A3, 0)) ---- [7 8 9]

print(np.max(A3, 1)) ---- [3 6 9]

Array - A1

Axis =0				
1	2	3	4	5

Array - A2

Axis	0		
1	1	2	3
	4	5	6

Array - A3

Axis	0		
1	1	2	3
	4	5	6
	7	8	9

4. argmax() – find the index of max elements in array.

import numpy as np

A1=np.array([1,2,3,4,5])

A2=np.array([[1,2,3], [4,5,6]])

A3=np.array([[1,2,3], [4,5,6] , [7,8,9]])

#1D Array – A1

print(np.argmax(A1)) ---- 4

#2D Array – A2

```
print( np.argmax(A2))      ---- 5
print(np.argmax(A2, 0))    ---- [ 1  1  1 ]
print(np.argmax(A2, 1))    ---- [ 2  2 ]
```

#2D Array – A3

```
print( np.argmax(A3))      ---- 8
print(np.argmax(A3, 0))    ---- [ 2  2  2 ]
print(np.argmax(A3, 1 ))   ---- [ 2  2  2 ]
```

Array – A1

Axis =0				
1	2	3	4	5

Array – A2

Axis	0		
1	1	2	3
	4	5	6

Array – A3

Axis	0		
1	1	2	3
	4	5	6
	7	8	9

5. . mean() – find the mean of elements in array.

import numpy as np

A1=np.array([1,2,3,4,5])

A2=np.array([[1,2,3], [4,5,6]])

A3=np.array([[1,2,3], [4,5,6] , [7,8,9]])

#1D Array – A1

```
print( np.mean(A1))      ---- 3.0
```

#2D Array – A2

```
print( np.mean(A2))      ---- 3.5
print(np.mean(A2, 0))    ---- [ 2.5  3.5  4.5 ]
print(np.mean(A2, 1))    ---- [ 2.   5. ]
```

#2D Array – A3

```
print( np.mean(A3))      ---- 5.0
print(np.mean(A3, 0))    ---- [ 4.  5.  6. ]
print(np.mean(A3, 1 ))   ---- [2.  5.  8. ]
```


Array - A1

Axis =0				
1	2	3	4	5

Array - A2

Axis	0		
1	1	2	3
	4	5	6

Array - A3

Axis	0		
1	1	2	3
	4	5	6
	7	8	9

6. . median () – find the median of elements in array.

import numpy as np

A1=np.array([1,2,3,4,5])

A2=np.array([[1,2,3], [4,5,6]])

A3=np.array([[1,2,3], [4,5,6] , [7,8,9]])

#1D Array – A1

print(np.median(A1)) ---- 3.0

#2D Array – A2

print(np.median(A2)) ---- 3.5

print(np.median(A2, 0)) ---- [2.5 3.5 4.5]

print(np.median(A2, 1)) ---- [2. 5.]

#2D Array – A3

print(np.median(A3)) ---- 5.0

print(np.median(A3, 0)) ---- [4. 5. 6.]

print(np.median(A3, 1)) ---- [2. 5. 8.]

Array - A1

Axis =0				
1	2	3	4	5

Array - A2

Axis	0		
1	1	2	3
	4	5	6

Array - A3

Axis	0		
1	1	2	3
	4	5	6
	7	8	9

7. std() – find the standard deviation of array.

import numpy as np

A1=np.array([1,2,3,4,5])

A2=np.array([[1,2,3], [4,5,6]])

A3=np.array([[1,2,3], [4,5,6] , [7,8,9]])

#1D Array – A1

print(np.std(A1)) ---- 1.414

#2D Array – A2

print(np.std(A2)) ---- 1.707

print(np.std(A2, 0)) ---- [1.5 1.5 1.5]

print(np.std(A2, 1)) ---- [0.816 0.816]

#2D Array – A3

print(np.std(A3)) ---- 2.5819

print(np.std(A3, 0)) ---- [2.449 2.449 2.449]

print(np.std(A3, 1)) ---- [0.816 0.816 0.816]

Array – A1

Axis =0				
1	2	3	4	5

Array – A2

Axis	0		
1	1	2	3
	4	5	6

Array – A3

Axis	0		
1	1	2	3
	4	5	6
	7	8	9

7. variance() – find the variance of elements in array. It is equal to square root of standard deviation.

import numpy as np

A1=np.array([1,2,3,4,5])

A2=np.array([[1,2,3], [4,5,6]])

A3=np.array([[1,2,3], [4,5,6] , [7,8,9]])

#1D Array – A1

print(np.var(A1)) ---- 2.0

#2D Array – A2

print(np.var(A2)) ---- 2.916

print(np.var(A2, 0)) ---- [2.25 2.25 2.25]

print(np.var(A2, 1)) ---- [0.666 0.666]

#2D Array – A3

print(np.var(A3)) ---- 6.666

print(np.var(A3, 0)) ---- [6. 6. 6.]

print(np.var(A3, 1)) ---- [0.666 0.666 0.666]

Array - A1

Axis =0				
1	2	3	4	5

Array - A2

Axis	0		
1	1	2	3
	4	5	6

Array - A3

Axis	0		
1	1	2	3
	4	5	6
	7	8	9

Day-12:

Q.1 Multiple Type Question

1. Which one is not true for NumPy?
 - a. NumPy created by Travis Oliphant.
 - b. NumPy has ndarray object.
 - c. NumPy has many inbuilt functions for matrix manipulation.
 - d. NumPy is not available under open source project.

2. Which one is not true for accessing array in NumPy ndarray Object?
 - a. Slicing is not allowed.
 - b. Negative indexing allowed for arrays.
 - c. -1 is the index of last value in 1D array.
 - d. 0 is the index of first value in 1D array.

3. Which one is not true for Aggregate Function Dictionary?
 - a. Sum() – find the sum of array.
 - b. Min() – Find the minimum of array.
 - c. Argmin() – find the index of minimum from array.
 - d. Avg() – find the average of the value in array.

4. Which one is not true Creating Matrix?
 - a. Empty() - create an array of specified shape and size.
 - b. Ones() - create an array filled with ones.
 - c. Zeros() – create an array filled with zeros.
 - d. NonEmpty()- create an initialized array.

5. Which is not valid for array?
 - a. Ndim – shows the dimension of the array.
 - b. Axis=0 represent the column of the 2D array.
 - c. Axis=1 represent the rows of the 2D array.
 - d. Row- shows the rows of the array.

Q.2 What is the output ?

1. <code>print(np.__version__)</code> <code>np.show_config()</code>	2. <code>Z = np.zeros(10)</code> <code>print(Z)</code>	3. <code>Z = np.zeros((10,10))</code> <code>print("%d bytes" % (Z.size * Z.itemsize))</code>
4. <code>nz =</code> <code>np.nonzero([1,2,0,0,4,0])</code> <code>print(nz)</code>	5. <code>Z = np.eye(3)</code> <code>print(Z)</code>	6. <code>Z = np.random.random(30)</code> <code>m = Z.mean()</code> <code>print(Z)</code> <code>print(m)</code>
7. <code>Z = np.ones((10,10))</code> <code>Z[1:-1,1:-1] = 0</code> <code>print(Z)</code>	8. <code>Z = np.zeros(10)</code> <code>Z[4] = 1</code> <code>print(Z)</code>	9. <code>x=np.ones((3,3), dtype=bool)</code> <code>print(x)</code>

Q.2 What is the output ?

1. <code>import numpy as np</code> <code>my_array = np.array([1, 2, 3, 4])</code> <code>my_slice = my_array[1:3]</code> <code>print(my_slice)</code>	2. <code>my_vector = np.array([1, 2, 3, 4, 5, 6])</code> <code>selection = my_vector % 2 == 0</code> <code>my_vector[selection]</code>
3. <code>import numpy as np</code> <code>my_array = np.array([1, 2, 3])</code> <code>my_slice = my_array[1:3]</code> <code>my_slice = my_slice * 2</code> <code>print(my_slice)</code>	4. <code>import numpy as np</code> <code>my_array = np.array([[1, 2, 3], [4, 5, 6]])</code> <code>my_slice = my_array[:, 1:3]</code> <code>print(my_slice)</code>
5. <code>import numpy as np</code> <code>first_matrix = np.array([[1, 2, 3], [4, 5, 6]])</code> <code>second_matrix = np.array([1, 2, 3])</code> <code>my_matrix=first_matrix + second_matrix</code> <code>print(my_matrix)</code>	6. <code>import numpy as np</code> <code>X = np.array([[[1, 2,3],</code> <code>[4, 5, 6]],</code> <code>[[7,8,9],</code> <code>[10,11,12]]])</code> <code>print(X.shape)</code> <code>print(X.ndim)</code> <code>print(X.size)</code>

Q.3 Find the output based on this data

Sales data of Parle Biscuit(in KG)

Name	Jan	Feb	Mar	Apr	May	Jun	
Ajay	10	21	23	31	7	22	
Vijay	13	17	12	29	14	16	
Sanjay	17	15	16	13	18	10	
Ajit	45	21	7	34	22	34	
Vikas	22	56	76	34	22	16	
Vipul	12	17	22	36	31	23	
Rakesh	31	23	27	41	32	22	

1. Find the Total Sales of each sales person.
2. Find the Maximum Sales of each sales person
3. Show the sales data of “Vikas” for all the months
4. Show the sales data of “Ajit” for the month of Apr, May, Jun.
5. Show the Sales data of all the sales person for the month of APR.
6. Show the sales data of FEB and JUN for the entire sales person.
7. Find average sales of each sales person.
8. Find the Minimum sales of each sales person.
9. Find the Minimum sales for each month.

Q.4

N= [1 , 2 , 3 , 4, 5]

A] Create a Numpy array

B] Write a lambda function to calculate [use MAP function]

$$F(x) - 3*x*x + 5*x + 10$$

C) Write a function to calculate the factorial of each number. [use MAP function]

NumPy Data Types

Python provides the following data types :-

- **strings** - text data enclosed in quote marks. eg. "ABCD"
- **integer** - integer numbers. eg. 1,2,3,-1, -2, -3
- **float** - real numbers. eg. 12.2, 12.42
- **boolean** - True or False.
- **complex** - a number in complex form. eg. $2.0 + 2.0j$, $2.5 + 1.5j$

NumPy provides more data types, and referenced by one character

- i – integer where (i1 = int8 , i2=int16, i3=int32 , i4=int64)
- b - boolean
- u - unsigned integer
- f - float
- c - complex float
- m - timedelta
- M - datetime
- O - object
- S - string
- U - unicode string
- V - fixed chunk of memory for other type (void)

NumPy Data Types.

dtype

- NumPy array has property **dtype**.
- **dtype** can be used to define the data type of the array , while creating.
- **dtype** also returns the data type of the array elements.

Example

- `a = np.array([1, 2, 3,4,5])`
- `a = np.array([1, 2, 3,4,5], dtype='i2')`
- `a = np.array([1, 2, 3,4,5], dtype='S')`
- `a = np.array([1, 2, 3,4,5], dtype='f')`
- `print('Data Type of Array: ', a.dtype)`

astype()

1. NumPy array has inbuilt method `astype()`.
2. `astype()` allows to convert the data type of an existing array, to the specified data type.

Example

```
a = np.array([1.1, 2.5, 0.6, 4.7, 5.3])
a=a.astype("i")
a=a.astype(bool)
[1.1 2.5 3.6 4.7 5.3]
[1 2 3 4 5]
[ True  True False  True  True]
```

Example for dtype()

```
import numpy as np
a = np.array([1, 2, 3,4,5])
print('Values of Array')
print(a)
print('Data Type of Array: ', a.dtype)
a = np.array([1, 2, 3,4,5], dtype='i2')
print('Values of Array')
print(a)
print('Data Type of Array: ', a.dtype)
a = np.array([1, 2, 3,4,5], dtype='S')
print('Values of Array')
print(a)
print('Data Type of Array: ', a.dtype)
a = np.array([1, 2, 3,4,5], dtype='f' )
print('Values of Array')
print(a)
print('Data Type of Array: ', a.dtype)
```

Output

```
Values of Array
[1 2 3 4 5]
Data Type of Array: int32
```

```
Values of Array
[1 2 3 4 5]
```


Data Type of Array: int16

Values of Array

[b'1' b'2' b'3' b'4' b'5']

Data Type of Array: |S1

Values of Array

[1. 2. 3. 4. 5.]

Data Type of Array: float32

Example for astype()

```
import numpy as np
```

```
a = np.array([1.1, 2.5, 0.6,4.7,5.3])
```

```
print('Values of Array')
```

```
print(a)
```

```
print('Data Type of Array: ', a.dtype)
```

```
a=a.astype("i")
```

```
print('Values of Array')
```

```
print(a)
```

```
print('Data Type of Array: ', a.dtype)
```

```
a=a.astype(bool)
```

```
print('Values of Array')
```

```
print(a)
```

```
print('Data Type of Array: ', a.dtype)
```

Output

Values of Array

[1.1 2.5 3.6 4.7 5.3]

Data Type of Array: float64

Values of Array

[1 2 3 4 5]

Data Type of Array: int32

Values of Array

[True True False True True]

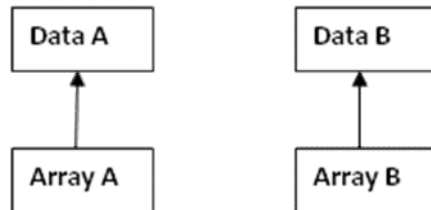
Data Type of Array: bool

NumPy - Copy() , View() and base

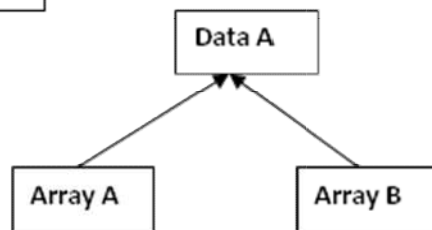
1. In case of **copy()** method, copied data has no link to the original array and changes does not reflected in the original base array.

2. In case of **view()** method, copied data has link to the original array and changes reflected in the original base array.
3. NumPy array has the property **base** , that returns the link to the original array. e.g., if array copied with **view()**.

Copy()



View()



```
# Example for copy() , view()
import numpy as np
a = np.array( [1.1, 2.5, 0.6, 4.7, 5.3] )
print('Values of Array - A ')
print(a)
print('Base of Array-A: ', a.base )
b=a.copy()
b[0]=10;
print('Values of Array -A ')
print(b)
print('Base of Array-B: ', b.base )
c=a.view()
c[0]=10;
print('Values of Array -A ')
print(c)
print('Base of Array-C: ', c.base )
```

Output

```
Values of Array - A
[1.1 2.5 0.6 4.7 5.3]
```

Base of Array-A: None

Values of Array -A

[10. 2.5 0.6 4.7 5.3]

Base of Array-B: None

Values of Array -A

[10. 2.5 0.6 4.7 5.3]

Base of Array-C: [10. 2.5 0.6 4.7 5.3]

NumPy- Shape & Reshape

- **Shape and Reshape**
- NumPy array has property “**shape**” which returns the tuple with count of elements in each dimension.
- For example – (2,4) means array has 2 rows , 4 column (values in each rows).
- NumPy array has inbuilt method “**reshape()**” used to change the dimension of array.
- **Example**
- `a = np.array([1, 2, 3, 4,5, 6, 7, 8])` --- shape - (8,) Row -8 , Column-0
- `a= a.reshape(2,4)` --- shape - (2, 4) Row -2 , Column-4
- `a = np.array([[1, 2, 3,4], [5, 6, 7, 8]])` --- shape - (2, 4) Row -2 , Column-4
- `a= a.reshape(8)` --- shape - (8,) Row -8 , Column-0
-

Example-1

```
import numpy as np
a = np.array([1, 2, 3, 4,5, 6, 7, 8])
print('Array Before : ')
print(a)
print('shape of array :', a.shape)
a=a.reshape(2,4)
print('Array After : ')
print(a)
print('shape of array :', a.shape)
a = np.array([[1, 2, 3,4], [5, 6, 7, 8]])
print('Array Before : ')
print( a)
a=a.reshape(8)
print('Array After : ')
print(a)
print('shape of array :', a.shape)
```

Output

Array Before :

```
[1 2 3 4 5 6 7 8]
```

shape of array : (8,)

Array After :

```
[[1 2 3 4]
```

```
[5 6 7 8]]
```

shape of array : (2, 4)

Array Before :

```
[[1 2 3 4]
```

```
[5 6 7 8]]
```

Array After :

```
[1 2 3 4 5 6 7 8]
```

shape of array : (8,)

NumPy- Array Iterating

- NumPy allows the iterating of array values using the for..loop.
- For 1-D Array for row in n :
 print(row)
- For 2-D Array for row in n :
 for col in row :
 print(col)
- For 3-D Array for z in n:
 for row in z :
 for col in row :
 print(col)
- `nditer()` method allows accessing each value in the array of 1-D, 2-D or 3-D.
 for x in np.nditer(n_arr) :
 print(x)
- `ndenumerate()` method returns the pair (index , value) for each element of the array.
 for id , x in np.ndenumerate(n_arr) :
 print(id , x)

Example

```
import numpy as np
```

```
a = np.array([ [1, 2, 3, 4],[5, 6, 7, 8]])
```

```

print('Simple Loop - Print Rows')
for row in a:
    print(row)

print('Simple Loop - Print each value')
for row in a:
    for col in row:
        print(col)

print('nditer( ) - Print each value')
for val in np.nditer(a):
    print(val)

print('ndenumerate( ) - Print (id ,value)')
for id,val in np.ndenumerate(a):
    print(id,val)

```

Output

Simple Loop - Print Rows

[1 2 3 4]

[5 6 7 8]

Simple Loop - Print each value

1 2 3 4 5 6 7 8

nditer() - Print each value

1 2 3 4 5 6 7 8

ndenumerate() - Print (id ,value)

(0, 0) 1

(0, 1) 2

(0, 2) 3

(0, 3) 4

(1, 0) 5

(1, 1) 6

(1, 2) 7

(1, 3) 8

NumPy- Filtering , Searching & Sorting

- `numpy.where()` is used to search the value in the array and it return the tuple of index.
- **Example**
 - `np.where(arr==4)`
 - `np.where(arr%2 ==0)`

- Filtering is the process of extracting some value from the array based on certain conditions.
- **Example**
 - `arr=[10,11,12,13,14]`
 - `filter= (arr%2 ==0)`
 - `arr=arr[filter]`
- `Numpy.sort(array)` is used to arrange the value of the array passed.
- **Example**
`np.sort(arr)`

Example

```
import numpy as np
a = np.array([ 1,4,3,2,5,8,7,6])
print('Values of Array - A ')
print(a)
print('Sorted Array: ')
print(np.sort(a))
print('Search the Even Numbers')
x=np.where(a%2==0)
print(x)
```

Output

```
Values of Array - A
[1 4 3 2 5 8 7 6]
Sorted Array:
[1 2 3 4 5 6 7 8]
Search the Even Numbers
(array([1, 3, 5, 7], dtype=int32),)
```

#Example for Searching

```
t = np.array([ [10,20,30,'red'],
               [21,31,41,'green'],
               [50,60,70,'brown'],
               [89,97,11,'green']  ])
row= np.where(t == 'green')
print(row)
rs= t[row]
print(rs)
```

Output

```
(array([1, 3], dtype=int32), array([3, 3], dtype=int32))
['green' 'green']
```

NumPy- Searching in 2D Array

Example

```
import numpy as np
a = np.array([ [1,5,2,5], [5,7,6,5] ])
print('Searching the Value')
filter=np.where(a==5)
print(filter)
x=a[filter]
print(x)
t = np.array([[10,20,30,'red'],
[21,'green',41,'green'],
[50,60,70,'brown'],
[89,97,11,'green']])
filter=np.where(t == 'green')
print(filter)
x=t[filter]
print(x)
```

Output

Searching ing the Value

```
(array( [0, 0, 1, 1], dtype=int32), array( [1, 3, 0, 3], dtype=int32))
[5 5 5 5]
(array( [1, 1, 3], dtype=int32), array( [1, 3, 3], dtype=int32))
['green' 'green' 'green']
```

Values found at this index

0 0 1 1	1 1 3
1 3 0 3	1 3 3
(0,1) (0,3) (1,0) (1,3)	(1,1) (1,3) (3,3)

NumPy- Filtering

Example

```
import numpy as np
a = np.array([ [1,5,2,5], [5,7,6,5] ])
print('Filtering the Value')
filter=(a%2==0)
print(filter)
x=a[filter]
print(x)
t = np.array([[10,20,30,'red'],
[21,31,41,'green'],
[50,60,70,'brown'],
[89,97,11,'green']])
filter=(t == 'green')
print(filter)
```

```
x=t[filter]
print(x)
```

Output

Filtering the Value

```
[ [False False True False]
  [False False True False] ]
```

```
[2 6]
```

```
[ [False False False False]
  [False False False True]
  [False False False False]
  [False False False True] ]
```

```
['green' 'green']
```

NumPy - Fancy Indexing

```
>>> a[(0,1,2,3,4), (1,2,3,4,5)]
array([1, 12, 23, 34, 45])

>>> a[3:, [0,2,5]]
array([[30, 32, 35],
       [40, 42, 45],
       [50, 52, 55]])

>>> mask = np.array([1,0,1,0,0,1], dtype=bool)
>>> a[mask, 2]
array([2, 22, 52])
```

0	1	2	3	4	5
10	11	12	13	14	15
20	21	22	23	24	25
30	31	32	33	34	35
40	41	42	43	44	45
50	51	52	53	54	55

NumPy - Joining and Splitting

Joining Methods

- `.concatenate()` used to join to 2 arrays at the provided axis level. Default axis=0.

- `.hstack()` used to horizontally join 2 arrays
- `.vstack()` used to vertically join 2 arrays.

Splitting Methods

- `numpy.hsplit()` split the array horizontally.
- `numpy.vsplit()` split the array vertically.
- `numpy.array_split(array, count, axis=0)` split the array into specified number.

NumPy- Array Joining

Example

```
import numpy as np
a=np.array([ [1,2,3],[4,5,6]])
b=np.array([[11,12,13],[14,15,16]])
c=np.concatenate( (a,b), axis=0)
print('Array join at Axis=0')
print(c)
c=np.concatenate( (a,b), axis=1)
print('Array join at Axis=0')
print(c)
c=np.vstack((a,b))
print('Array join vertically')
print(c)
c=np.hstack( (a,b))
print('Array join Horizontally')
print(c)
```

Output

Array join at Axis=0

```
[ [ 1  2  3]
  [ 4  5  6]
  [11 12 13]
  [14 15 16] ]
```

Array join at Axis=0

```
[ [ 1  2  3 11 12 13]
  [ 4  5  6 14 15 16] ]
```

Array join vertically

```
[ [ 1  2  3]
  [ 4  5  6]
  [11 12 13]
  [14 15 16] ]
```

Array join Horizontally

```
[ [ 1 2 3 11 12 13]
  [ 4 5 6 14 15 16] ]
```

NumPy- Array Splitting

```
import numpy as np
a=np.array([ [1,2,3],[4,5,6], [11,12,13], [14,15,16] ] )
c=np.array_split( a, 2)
print('Array Split into 2 part')
print(c)
c=np.array_split( a, 3, axis=0)
print('Array Split at Axis=0')
print(c)
c=np.array_split( a, 3, axis=1)
print('Array Split at Axis=1')
print(c)
c=np.hsplit( a,3)
print('Array Split Horizontally')
print(c)
c=np.vsplit(a,2)
print('Array Split vertically')
print(c)
```

Output

Array Split into 2 part

```
[array([[1, 2, 3],
        [4, 5, 6]]), array([[11, 12, 13],
        [14, 15, 16]])]
```

Array Split at Axis=0

```
[array([[1, 2, 3],
        [4, 5, 6]]), array([[11, 12, 13]]), array([[14, 15, 16]])]
```

Array Split at Axis=1

```
[array([[ 1], [ 4], [11], [14]]), array([[ 2], [ 5], [12], [15]]), array([[ 3], [ 6], [13], [16]])]
```

Array Split Horizontally

```
[array([[ 1], [ 4], [11], [14]]), array([[ 2], [ 5], [12], [15]]), array([[ 3], [ 6], [13], [16]])]
```

Array Split vertically

```
[array([[1, 2, 3], [4, 5, 6]]), array([[11, 12, 13], [14, 15, 16]])]
```

NumPy – arange()

- The advantage of `numpy.arange()` over the normal in-built `range()` function is that it allows us to generate sequences of numbers that are not integers.
- `arange([start,] stop[, step,][, dtype])` : Returns an array with evenly spaced elements as per the interval. The interval mentioned is half opened i.e. [Start, Stop)
- Parameters :
 - `start` : [optional] start of interval range. By default `start = 0`
 - `stop` : end of interval range
 - `step` : [optional] step size of interval. By default `step size = 1`,

Example

```
import numpy as np
print("A1\n", np.arange(4).reshape(2, 2), "\n")
print("A2\n", np.arange(4, 10), "\n")
print("A3\n", np.arange(4, 20, 3), "\n")
```

```
# Printing all numbers from 1 to 2 in steps of 0.1
print(("A4\n", np.arange(1, 2, 0.1))
```

Output

```
A1      [ [0 1]
          [2 3] ]
A2      [4 5 6 7 8 9]
A3      [ 4  7 10 13 16 19]
A4      [1.  1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9]
```

Day-13:

Q.1 Multiple Choice Question

1. Which one is not true for NumPy data type?
 - a. 'S'- used for string data type.
 - b. **dtype** used get the data type of array.
 - c. **astype()** used to convert the data type of array.
 - d. **'D'- used for date data type.**

2. Which one is not true for NumPy Array?
 - a. **Copy() – copied data has link with original array.**
 - b. View() – copied data has link with original array.
 - c. Shape – gives the tuple with count of row and column.
 - d. Reshape() – allows to modify the shape of array.

3. Which one is not true for Array Iterating in NumPy?
 - a. **nditer()** allows to access each element of the array.
 - b. **ndenumerate()** allows to access each element of the array.
 - c. **for..loop** can be used to sequence through each row and column.
 - d. **nditer()** gives the index and value of each element in array.

4. Which one is not true for Searching and Sorting of NumPy array?
 - a. **where()** – search the element and provide the tuple of index.
 - b. **sort()** - used to arrange the element of array.
 - c. **Fancy index** - used to filter the elements of the array based on masking.
 - d. **Filtering** – used to extract element based on slicing.

5. Which is not valid for Array Split and join?
 - a. **hstack()** - joins arrays horizontally.
 - b. **hsplit()** – splits arrays horizontally.
 - c. **array_split()** – split the array into based on count.
 - d. **concatenate()** - joins arrays horizontally only.

Q.2 Write a NumPy program to test whether none of the elements of a given array is zero.

```
x = np.array([1, 2, 3, 4])
print("Original array:")
print(x)
print("Test if none of the elements of the said array is zero:")
print(np.all(x))
x = np.array([0, 1, 2, 3])
print("Original array:")
print(x)
print("Test if none of the elements of the said array is zero:")
print(np.all(x))
```

Q.3 Write a NumPy program to test element-wise for positive or negative.

```
import numpy as np
a = np.array([1, 0, -1, -4])
print("Original array")
print(a)
print("Test element-wise for positive or negative infinity:")
print(a>=0)
```

Q.4 Write a NumPy program to create an array of 10 zeros,10 ones, 10 fives.

```
import numpy as np
array=np.zeros(10)
print("An array of 10 zeros:")
print(array)
array=np.ones(10)
print("An array of 10 ones:")
print(array)
array=np.ones(10)*5
print("An array of 10 fives:")
print(array)
```

Q.5 Write a NumPy program to create a 10x10 matrix, in which the elements on the borders will be equal to 1, and inside 0.

```
import numpy as np
x = np.ones((10, 10))
x[1:-1, 1:-1] = 0
print(x)
```

Q.6 Write a NumPy program to extract the numbers divisible by 3 and 5 from a 2D Array.

```
import numpy as np
array=np.random.randint(100 , size=(3,3))

print("An array of 3 X 3 zeros:")
print(array)
print("An array of divisible by 3 and 5:")
filter= (array%3==0) & (array%5==0 )
print( array[ filter])
```

Q.7 Write a NumPy program to replace even number by 0 in 2D array.

```
import numpy as np
array=np.random.randint(100 , size=(3,3))
print("An array of 3 X 3 zeros:")
print(array)
print("An array of divisible by 3 and 5:")
filter= (array%2==0)
array[ filter]=0
print(array)
```

Q.8 What is the Output ?

```
x = np.random.randint(100, size=5)
print(x)
x1 = np.random.rand(5)
print(x1)
x2= np.random.choice([3, 5, 7, 9], size=5)
print(x2)
```

Q.9 What is the output

```
is_prime = np.ones((100,), dtype=bool)

# Cross out 0 and 1 which are not primes:
is_prime[:2] = 0

# cross out its higher multiples (sieve of Eratosthenes):
nmax = int(np.sqrt(len(is_prime)))
for i in range(2, nmax):
    is_prime[2*i::i] = False

print(np.nonzero(is_prime))
```

Q.10 Write a NumPy program to create an array with values ranging from 12 to 38.

Q.11 Write a NumPy program to create a 3x3 matrix with values ranging from 2 to 10.

[use arange and reshape]

Q. 12 Extract from the array `np.array([3,4,6,10,24,89,45,43,46,99,100])` with Boolean masking all the number

- which are not divisible by 3 `A [A%3 !=0]`
- which are divisible by 5 `A[A%5==0]`
- which are divisible by 3 and 5 `A [A%3==0 & A%5==0]`
- which are divisible by 3 and set them to 42 `A[A%3==0] =42`

Q.13 Write a NumPy program to read 2 matrixes of 3 X 4 dimensions and print the sum of corresponding element of each matrix.

Q.14 Find the output based on this data

Sales data of Parle Biscuit(in KG)

Sales Data for the month of Jan-Jun.

Name	Jan	Feb	Mar	Apr	May	Jun
Ajay	10	21	23	31	7	22
Vijay	13	17	12	29	14	16
Sanjay	17	15	16	13	18	10
Ajit	45	21	7	34	22	34
Vikas	22	56	76	34	22	16
Vipul	12	17	22	36	31	23
Rakesh	31	86	27	41	32	22

Sales Data for the month of Jul-Dec.

Name	Jul	Aug	Sep	Oct	Nov	Dec
Ajay	20	31	33	41	17	32
Vijay	23	27	86	39	24	26
Sanjay	27	25	26	23	28	20
Ajit	55	31	17	44	32	44
Vikas	32	66	86	44	32	26
Vipul	22	27	32	46	41	33
Rakesh	41	33	37	51	42	32

Perform the following operation on the data.

1. Find the Total Sales of each sales person.
2. Find the Maximum Sales of each sales person
3. Show the sales data of “Vikas” for all the months
4. Show the sales data of “Ajit” for the month of Apr, May, Jun.
5. Show the Sales data of all the sales person for the month of APR, AUG and DEC.
6. Find average sales of each sales person.
7. Find the Minimum sales of each sales person.
8. Find the Minimum sales for each month.
9. Find the Sales which are more than the average of sales data.
10. Find the Names of sales person who have achieved level of maximum sales in a year.

Steps to be followed

- 1) Read data from CSV file Saledata1-6.csv in SALE1, by using the
`np.genfromtxt('Saledata1-6.csv', delimiter=',', dtype='str')`
- 2) Read data from CSV file Saledata7-12.csv in SALE2, by using the above method.
- 3) Join the data using, **`SALE=np.hstack([SALE1, SALE2])`**.
- 4) After this you get multiple NAME column from the two files. Now delete this common Column name by using

`SALE=np.delete(SALE, 7 , 1)`

7 is the column number and 1 is the Axis { represent Column }

Using of Pandas

Example-1

```
import pandas as pd
S = pd.Series( [ 11, 28, 72, 3, 5, 8] )
print(S)
print(S.index)
print(S.values)
Output
0    11
1    28
2    72
3     3
4     5
5     8
dtype: int64
RangeIndex(start=0, stop=6, step=1)
[11 28 72  3  5  8]
```

Example-2

```
import pandas as pd
fruits = ['apple', 'orange', 'mango', 'pear']
quantities = [20, 30, 52, 15]
S = pd.Series(quantities, index=fruits)
print(S)
```

```
Output
apple    20
orange   30
mango    52
pear     15
dtype: int64
```

Example-3

```
import pandas as pd
data = {
    'Fruits':['apple', 'orange', 'mango', 'pear'],
    'Quantities':[20,30,52,15] }
df = pd.DataFrame(data)
print(df)
```

```
Output
   Fruits  Quantities
0  apple         20
1  orange         30
```

```
2 mango      52
3 pear       15
```

Data Structures

Pandas deals with the following two data structures –

- Series
- DataFrame

These data structures are built on top of Numpy array, which means they are fast.

Data Structure	Dimensions	Description
Series	1	1-D labelled homogeneous array. size immutable.
Data Frames	2	2-D labelled heterogeneous tabular structure . size-mutable

Python Pandas - Series

Series is a one-dimensional labelled array capable of holding data of any type (integer, string, float, python objects, etc.). The axis labels are collectively called index.

A pandas Series can be created using the following constructor –
`pandas.Series(data, index, dtype, copy)`

The parameters of the constructor are as follows –

1. Data: data takes various forms like ndarray, list, constants
2. Index: Index values must be unique and hashable, same length as data. Default `np.arange(n)` if no index is passed.
3. Dtype: dtype is for data type. If None, data type will be inferred
4. Copy: Copy data. Default False

Create an Empty Series

A basic series, which can be created is an Empty Series.

```
#import the pandas library and aliasing as pd
import pandas as pd
s = pd.Series( )
print(s)
```

Its output is as follows –
Series([], dtype: float64)

Create a Series from ndarray

1. If data is an ndarray, then index passed must be of the same length.
2. If no index is passed, then by default index will be range(n) where n is array length, i.e., [0,1,2,3.... range(len(array))-1].

Example

```
#import the pandas library and aliasing as pd
import pandas as pd
import numpy as np
data = np.array( ['a','b','c','d'] )
s = pd.Series(data)
print(s)
```

Create a Series from dict

1. A dict can be passed as input and if no index is specified, then the dictionary keys are taken in a sorted order to construct index.
2. If index is passed, the values in data corresponding to the labels in the index will be pulled out.

Example

```
#import the pandas library and aliasing as pd
import pandas as pd
data = { 'a': 0., 'b': 1., 'c': 2.}
s = pd.Series(data)
print(s)
```

Output

```
a    0.0
b    1.0
c    2.0
dtype: float64
```

Create a Series from Scalar

1. If data is a scalar value, an index must be provided.
2. The value will be repeated to match the length of index.

```
#import the pandas library and aliasing as pd
import pandas as pd
s = pd.Series(5, index=[0, 1, 2, 3])
print(s)
```

Output

```
0    5
1    5
```

```
2 5
3 5
dtype: int64
```

Accessing Data from Series with Position

1. Data in the series can be accessed similar to that in an ndarray.
2. Retrieve the element by providing index. The Index starts from zero for the array, which means the first element is stored at zeroth position and so on.

Example

```
import pandas as pd
s = pd.Series([1,2,3,4,5],index = ['a','b','c','d','e'])
#retrieve the first element
print(s)
print('By numeric index :', s[0])
print('By named index :', s['a'])
```

Output

```
a    1
b    2
c    3
d    4
e    5
dtype: int64
By numeric index : 1
By named index : 1
```

Retrieve Data Using Label (Index)

A Series is like a fixed-size dict in that you can get and set values by index label.

```
import pandas as pd
s = pd.Series([1,2,3,4,5],index = ['a', 'b', 'c', 'd', 'e'])

#retrieve a single element
print(s['a'])
```

DataFrame

A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns.

Features of DataFrame

- Potentially columns are of different types
- Size – Mutable
- Labeled axes (rows and columns)
- Can Perform Arithmetic operations on rows and columns

Structure

- We are creating a data frame with student's data.
- You can think of it as an SQL table or a spreadsheet data representation.

The diagram illustrates the structure of a DataFrame. It features a table with three columns: 'Regd. No', 'Name', and 'Marks%'. The rows are labeled with student IDs: 1000, 1001, 1002, 1003, and 1004. Arrows point from the labels 'Columns' and 'ROWS' to their respective axes in the table.

Regd. No	Name	Marks%
1000	Steve	86.29
1001	Mathew	91.63
1002	Jose	72.90
1003	Patty	69.23
1004	Vin	88.30

A pandas DataFrame can be created using the following constructor –
`pandas.DataFrame(data, index, columns, dtype, copy)`

The parameters of the constructor are as follows –

1. Data - Data takes various forms like ndarray, series, map, lists, dict, constants and also another DataFrame.
2. Index - For the row labels, the Index to be used for the resulting frame is Optional.
Default `np.arange(n)`, if no index is passed.
3. Columns- It is optional. Default is - `np.arange(n)`, if no index is passed.
4. Dtype- Data type of each column.
5. Copy- This command is used for copying of data, if the default is False.

Create DataFrame

A pandas DataFrame can be created using various inputs like –

- Lists

- dict
- Series
- Numpy ndarrays
- Another DataFrame

Create an Empty DataFrame

- A basic DataFrame, which can be created, is an Empty Dataframe.

```
#import the pandas library and aliasing as pd
import pandas as pd
df = pd.DataFrame( )
print(df)
```

Output
Empty DataFrame
Columns: []
Index: []

Create a DataFrame from Lists

The DataFrame can be created using a single list or a list of lists.

```
import pandas as pd
data = [1,2,3,4,5]
df = pd.DataFrame(data)
print(df)
```

Output

```
0 1
1 2
2 3
3 4
4 5
```

Create a DataFrame from Dict of ndarrays / Lists

1. All the ndarrays must be of same length. If index is passed, then the length of the index should equal to the length of the arrays.
2. If no index is passed, then by default, index will be range(n), where n is the array length.

Example

```
import pandas as pd
data = {'Name':['Tom', 'Jack', 'Steve', 'Ricky'], 'Age':[28,34,29,42]}
df = pd.DataFrame(data)
print(df)
```

Output

	Name	Age
0	Tom	28
1	Jack	34
2	Steve	29
3	Ricky	42

Create a DataFrame from List of Dicts

1. List of Dictionaries can be passed as input data to create a DataFrame.
2. The dictionary keys are by default taken as column names.
3. The following example shows how to create a DataFrame by passing a list of dictionaries.

```
import pandas as pd
data = [{'a': 1, 'b': 2}, {'a': 5, 'b': 10, 'c': 20}]
df = pd.DataFrame(data)
print(df)
```

Output

	a	b	c
0	1	2	NaN
1	5	10	20.0

Create a DataFrame from Dict of Series

1. Dictionary of Series can be passed to form a DataFrame.
2. The resultant index is the union of all the series indexes passed.

Example

```
import pandas as pd
d = { 'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
      'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd']) }
df = pd.DataFrame(d)
print(df)
```

Output

	one	two
a	1.0	1
b	2.0	2
c	3.0	3
d	NaN	4

Column Selection

Selecting a column from the DataFrame.

```
import pandas as pd
d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
      'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}
df = pd.DataFrame(d)
print(df['one'])
```

Output

```
a    1.0
b    2.0
c    3.0
d    NaN
Name: one, dtype: float64
```

Column Addition

```
import pandas as pd
d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
      'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}
df = pd.DataFrame(d)
Adding a new column to an existing DataFrame
print ("Adding a new column by passing as Series:")
df['three']=pd.Series([10,20,30], index=['a', 'b', 'c'])
print(df)
print ("Adding a new column using the existing columns in DataFrame:")
df['four']=df['one']+df['three']
print(df)
```

Column Deletion - Columns can be deleted or popped.

Example

```
import pandas as pd
d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
      'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd']),
      'three' : pd.Series([10,20,30], index=['a','b','c'])}
df = pd.DataFrame(d)
print("Our dataframe is:")
print(df)
print ("Deleting the first column using DEL function:")
del df['one']
print(df)
print ("Deleting another column using POP function:")
df.pop('two')
print(df)
```


Our dataframe is:

	one	two	three
a	1.0	1	10.0
b	2.0	2	20.0
c	3.0	3	30.0
d	NaN	4	NaN

Deleting the first column using DEL function:

	two	three
a	1	10.0
b	2	20.0
c	3	30.0
d	4	NaN

Deleting another column using POP function:

	three
a	10.0
b	20.0
c	30.0
d	NaN

Row Selection

1. Pandas provide a unique method to retrieve rows from a Data frame.
2. DataFrame.loc[] method is used to retrieve rows from Pandas DataFrame.
3. Rows can also be selected by passing integer location to an iloc[] function.

```
import pandas as pd
# making data frame from csv file
d = {'Name':['Tom', 'Jack', 'Steve', 'Ricky'], 'Age':[28,34,29,42]}
df = pd.DataFrame(d, index= ['e1','e2', 'e3','e4'])
print("Our dataframe is:")
print(df)
# retrieving row by loc method
first = df.loc['e1']
second = df.loc['e2']
print(first, "\n\n", second)
```

Our dataframe is:

	Name	Age
e1	Tom	28
e2	Jack	34
e3	Steve	29
e4	Ricky	42

```
Name    Tom
Age     28
Name: e1, dtype: object
```

```
Name    Jack
Age     34
Name: e2, dtype: object
```

Row Addition- In Order to add a Row in Pandas DataFrame, we can concat the old dataframe with new .

```
import pandas as pd
d = {'Name':['Tom', 'Jack', 'Steve', 'Ricky'], 'Age':[28,34,29,42]}
df = pd.DataFrame(d, index= ['e1','e2', 'e3','e4'])
print("Our dataframe is:")
print(df)
```

```
# simply concatenate both dataframes
new_row = pd.DataFrame({'Name':'Ajay', 'Age':33},index =[0] )
df = pd.concat( [new_row, df] )
print(df)
```

```
df = pd.concat( [new_row, df] ).reset_index(drop = True)
print(df)
```

Our dataframe is:

	Name	Age
e1	Tom	28
e2	Jack	34
e3	Steve	29
e4	Ricky	42

	Name	Age
0	Ajay	33
e1	Tom	28
e2	Jack	34
e3	Steve	29
e4	Ricky	42

	Name	Age
0	Ajay	33
1	Tom	28
2	Jack	34

```
3    Steve  29
4    Ricky  42
```

Row Deletion: In Order to delete a row in Pandas DataFrame, we can use the drop() method. Rows is deleted by dropping Rows by index label.

```
d = {'Name':['Tom', 'Jack', 'Steve', 'Ricky'], 'Age':[28,34,29,42]}
df = pd.DataFrame(d, index= ['e1','e2', 'e3','e4'])
print("Our dataframe is:")
print(df)
```

```
# dropping values
df.drop(['e1', 'e4'], inplace = True)
print(df)
```

Our dataframe is:

	Name	Age
e1	Tom	28
e2	Jack	34
e3	Steve	29
e4	Ricky	42

	Name	Age
e2	Jack	34
e3	Steve	29

Python Pandas - Sorting

Using the sort_index() method, by passing the axis arguments and the order of sorting, DataFrame can be sorted. By default, sorting is done on row labels in ascending order.

```
import pandas as pd
import numpy as np
unsorted_df = pd.DataFrame(np.random.randn(10,2), index=[1,4,6,2,3,5,9,8,0,7], columns =
['col2', 'col1'])
sorted_df=unsorted_df.sort_index()
print(sorted_df)
```

Unsorted

	col2	col1
1	0.402119	0.578492
4	1.511547	0.051883
6	-1.074378	0.009969
2	2.242691	1.241852
3	-1.078832	-1.272054
5	-1.062808	-2.144484
9	0.578359	0.320623
8	1.242311	-0.322329
0	-0.762395	-1.476712
7	1.486764	0.620362

Sorted

	col2	col1
0	-0.762395	-1.476712
1	0.402119	0.578492
2	2.242691	1.241852
3	-1.078832	-1.272054
4	1.511547	0.051883
5	-1.062808	-2.144484
6	-1.074378	0.009969
7	1.486764	0.620362
8	1.242311	-0.322329
9	0.578359	0.320623

Boolean Indexing in Pandas

1. In boolean indexing, we will select subsets of data based on the actual values of the data in the DataFrame and not on their row/column labels or integer locations.
2. In boolean indexing, we use a boolean vector to filter the data.
3. Boolean indexing is a type of indexing which uses actual values of the data in the DataFrame.
4. In boolean indexing, we can filter a data in four ways –
 1. Accessing a DataFrame with a boolean index
 2. Applying a boolean mask to a dataframe
 3. Masking data based on column value
 4. Masking data based on index value

Accessing a DataFrame with a boolean index

In order to access a dataframe with a boolean index, we have to create a dataframe in which index of dataframe contains a boolean value that is “True” or “False”.

Example

```
# importing pandas as pd
import pandas as pd
```

```
# dictionary of lists
```

```
dict = {'name':["aparna", "pankaj", "sudhir", "Geeku"],
        'degree': ["MBA", "BCA", "M.Tech", "MBA"],
        'score':[90, 40, 80, 98]}
```

```
df = pd.DataFrame(dict, index = [True, False, True, False])
print(df)
```

	name	degree	score
True	aparna	MBA	90
False	pankaj	BCA	40
True	sudhir	M.Tech	80
False	Geeku	MBA	98

1. Now we have created a dataframe with boolean index after that user can access a dataframe with the help of boolean index.
2. User can access a dataframe using three functions that is `.loc[]`, `.iloc[]`, `.ix[]`

Accessing a Dataframe with a boolean index using `.loc[]`

In order to access a dataframe with a boolean index using `.loc[]`, we simply pass a boolean value (True or False) in a `.loc[]` function.

```
import pandas as pd
dict = {'name':["aparna", "pankaj", "sudhir", "Geeku"],
        'degree': ["MBA", "BCA", "M.Tech", "MBA"],
        'score':[90, 40, 80, 98]}
# creating a dataframe with boolean index
df = pd.DataFrame(dict, index = [True, False, True, False])
# accessing a dataframe using .loc[] function
print(df.loc[True])
```

	name	degree	score
True	aparna	MBA	90
False	pankaj	BCA	40
True	sudhir	M.Tech	80
False	Geeku	MBA	98

	name	degree	score
True	aparna	MBA	90
True	sudhir	M.Tech	80

Accessing a Dataframe with a boolean index using .iloc[]

In order to access a dataframe using .iloc[], we have to pass a boolean value (True or False) in a .iloc[] function but .iloc[] function accept only integer as argument so it will throw an error so we can only access a dataframe when we pass a integer in .iloc[] function

```
import pandas as pd
dict = {'name':["aparna", "pankaj", "sudhir", "Geeku"],
        'degree': ["MBA", "BCA", "M.Tech", "MBA"],
        'score':[90, 40, 80, 98]}
df = pd.DataFrame(dict, index = [True, False, True, False])
print(df.iloc[True])
```

```
import pandas as pd
dict = {'name':["aparna", "pankaj", "sudhir", "Geeku"],
        'degree': ["MBA", "BCA", "M.Tech", "MBA"],
        'score':[90, 40, 80, 98]}
df = pd.DataFrame(dict, index = [True, False, True, False])
print(df.iloc[1])
```

```
name      pankaj
degree    BCA
score      40
dtype: object
```

Accessing a Dataframe with a boolean index using .ix[]

In order to access a dataframe using .ix[], we have to pass boolean value (True or False) and integer value to .ix[] function because as we know that .ix[] function is a hybrid of .loc[] and .iloc[] function.

```
import pandas as pd
dict = {'name':["aparna", "pankaj", "sudhir", "Geeku"],
        'degree': ["MBA", "BCA", "M.Tech", "MBA"],
        'score':[90, 40, 80, 98]}
df = pd.DataFrame(dict, index = [True, False, True, False])
print(df.ix[True])
```

	name	degree	score
True	aparna	MBA	90
True	sudhir	M.Tech	80

Accessing a DataFrame with a boolean index

```
# importing pandas as pd
import pandas as pd

# Dictionary of lists
dict = {'name':["aparna", "pankaj", "sudhir", "Geeku"],
        'degree': ["BCA", "BCA", "M.Tech", "BCA"],
        'score':[90, 40, 80, 98]}

# creating a dataframe
df = pd.DataFrame(dict)

# using a comparison operator for filtering of data
print(df['degree'] == 'BCA')
```

```
name      pankaj
degree    BCA
score      40
dtype: object
```

Day-14:

Q.1 Multiple Choice Question

1. Which one is not true for Pandas Module ?
 - a. It is free software library for Python.
 - b. It provides – Series and Dataframe data structures.
 - c. It does not have graphical capabilities.
 - d. It enhances the NumPy, Scipy and Matplotlib capabilities.
2. Which one is not true for Pandas Data Structure?
 - a. Series is a 1-D labeled homogeneous array.
 - b. Series allows accessing values by index.
 - c. Dataframe is a 2-D labeled heterogeneous array.
 - d. Dataframe allows accessing values as index in loc().
3. Which one is not true for Series?
 - a. Series can be created from dataframe.
 - b. Series can be created from dictionary.
 - c. Series allows accessing the items using integer index.
 - d. Series allows accessing the items using label index.
4. Which one is not true for Dataframe?
 - a. **df.pop('col')** - remove the column from the dataframe.
 - b. **del df['col']** - can be used to remove the column from the dataframe.
 - c. Dataframe cannot be created from dictionary of series.
 - d. Dataframe can be created from List of dictionary.
5. Which is not valid for Dataframe Accessing?
 - a. **df.loc[]** allows accessing rows based on Boolean index.
 - b. **df.loc[]** allows accessing rows based on Integer index .
 - c. **df.iloc[]** allows accessing rows based on Boolean index only.
 - d. **df.iloc[]** allows accessing rows based on integer index only.

Q.2 What is the output

```
import pandas as pd
df = pd.DataFrame({'X':[78,85,96,80,86], 'Y':[84,94,89,83,86], 'Z':[86,97,96,72,83]});
print(df)
```


Q.3 Perform the Following Operation on Data Frame

1. Create a dataframe with following data of month 1-6

month_number	facecream	facewash	toothpaste	bathingsoap	shampoo	moisturizer
1	2500	1500	5200	9200	1200	1500
2	2630	1200	5100	6100	2100	1200
3	2140	1340	4550	9550	3550	1340
4	3400	1130	5870	8870	1870	1130
5	3600	1740	4560	7760	1560	1740
6	2760	1555	4890	7490	1890	1555

2. Add the column Total_Sales.
3. Create a new data frame with data of month 7-11

7	2980	1120	4780	8980	1780	1120
8	3700	1400	5860	9960	2860	1400
9	3540	1780	6100	8100	2100	1780
10	1990	1890	8300	10300	2300	1890
11	2340	2100	7300	13300	2400	2100

4. Concatenate this data Frame with the DataFrame created in Step-1
5. Print the sales detail of month -7 and month-8.
6. Print the sales detail of shampoo.
7. Print the Total_Sales of all the months.

Q.4 Find the output based on this data

Sales data of Parle Biscuit(in KG)

Name	Jan	Feb	Mar	Apr	May	Jun
Ajay	10	21	23	31	7	22
Vijay	13	17	12	29	14	16
Sanjay	17	15	16	13	18	10
Ajit	45	21	7	34	22	34
Vikas	22	56	76	34	22	16
Vipul	12	17	22	36	31	23
Rakesh	31	23	27	41	32	22

1. Find the Total Sales of each sales person. [df.sum(axis=1)]
2. Find the Maximum Sales of each sales person [df.max(axis=1)]
3. Show the sales data of “Vikas” for all the months [df.loc[4]]
4. Show the Sales data of all the sales person for the month of APR. [df['Apr']]
5. Show the sales data of FEB and JUN for the entire sales person. [df[['Feb', 'Jun']]
6. Show the sales data of “Ajit” for the month of Apr, May, Jun.

```
[ n=df.loc[ 3]
      n[ ['Apr','May','Jun'] ]      ]
```

7. Find average sales of each sales person. [df.mean(axis=1)]
8. Find the Minimum sales of each sales person. [df.min(axis=1)]
9. Find the Minimum sales for each month. [df.min(axis=0)]
10. Find the median sales of each sales person. [df.median(axis=1)]
11. Find the median sales for each month. [df.median(axis=0)]

```
import pandas as pd
df=pd.read_csv("d:\\pyprg\\salesdata.csv")
print(df)
```

Working with Missing Data

In Pandas missing data is represented by two value:

- **None:** None is a Python singleton object that is often used for missing data in Python code.
- **NaN :** NaN (an acronym for Not a Number), is a special floating-point value recognized by all systems that use the standard IEEE floating-point representation

Pandas treat None and NaN as essentially interchangeable for indicating missing or null values. To facilitate this convention, there are several useful functions for detecting, removing, and replacing null values in Pandas DataFrame :

- `isnull()`
- `notnull()`
- `dropna()`
- `fillna()`
- `replace()`
- `interpolate()`

Checking for missing values using `isnull()`

```
# importing pandas as pd
import pandas as pd
```

```
# importing numpy as np
import numpy as np
```

```
# dictionary of lists
dict = {'First Score':[100, 90, np.nan, 95],
        'Second Score': [30, 45, 56, np.nan],
        'Third Score':[np.nan, 40, 80, 98]}
```

```
# creating a dataframe from list
df = pd.DataFrame(dict)
```

```
# using isnull() function
df.isnull()
```

Output:

	First Score	Second Score	Third Score
0	False	False	True
1	False	False	False
2	True	False	False
3	False	True	False

Checking for missing values using notnull()

```
# importing pandas as pd
import pandas as pd

# importing numpy as np
import numpy as np

# dictionary of lists
dict = {'First Score':[100, 90, np.nan, 95],
        'Second Score': [30, 45, 56, np.nan],
        'Third Score':[np.nan, 40, 80, 98]}

# creating a dataframe using dictionary
df = pd.DataFrame(dict)

# using notnull( ) function
df.notnull()
```

Output:

	First Score	Second Score	Third Score
0	True	True	False
1	True	True	True
2	False	True	True
3	True	False	True

Filling missing values using fillna()

```
# importing pandas as pd
import pandas as pd

# importing numpy as np
import numpy as np

# dictionary of lists
dict = {'First Score':[100, 90, np.nan, 95],
        'Second Score': [30, 45, 56, np.nan],
        'Third Score':[np.nan, 40, 80, 98]}

# creating a dataframe from dictionary
df = pd.DataFrame(dict)

# filling missing value using fillna()
df.fillna(0)
```

	First Score	Second Score	Third Score
0	100.0	30.0	0.0
1	90.0	45.0	40.0
2	0.0	56.0	80.0
3	95.0	0.0	98.0

Filling null values with the previous ones

```
# importing pandas as pd
import pandas as pd
```

```
# importing numpy as np
import numpy as np
```

```
# dictionary of lists
dict = {'First Score':[100, 90, np.nan, 95],
        'Second Score': [30, 45, 56, np.nan],
        'Third Score':[np.nan, 40, 80, 98]}
```

```
# creating a dataframe from dictionary
df = pd.DataFrame(dict)
```

```
# filling a missing value with
# previous ones
df.fillna(method='pad')
```

Output:

	First Score	Second Score	Third Score
0	100.0	30.0	NaN
1	90.0	45.0	40.0
2	90.0	56.0	80.0
3	95.0	56.0	98.0

Filling null value with the next ones

```
# importing pandas as pd
import pandas as pd

# importing numpy as np
import numpy as np

# dictionary of lists
dict = {'First Score':[100, 90, np.nan, 95],
        'Second Score': [30, 45, 56, np.nan],
        'Third Score':[np.nan, 40, 80, 98]}

# creating a dataframe from dictionary
df = pd.DataFrame(dict)

# filling null value using fillna() function
df.fillna(method='bfill')
```

Output:

	First Score	Second Score	Third Score
0	100.0	30.0	40.0
1	90.0	45.0	40.0
2	95.0	56.0	80.0
3	95.0	NaN	98.0

Filling a null values using replace() method

```
# importing pandas as pd
import pandas as pd

# importing numpy as np
import numpy as np
```

```
# dictionary of lists
dict = {'First Score':[100, 90, np.nan, 95],
        'Second Score': [30, 45, 56, np.nan],
        'Third Score':[np.nan, 40, 80, 98]}

# creating a dataframe from dictionary
df = pd.DataFrame(dict)
df.replace(to_replace = np.nan, value = -99,inplace=True)
print(df)
```

Output

	First Score	Second Score	Third Score
0	100.0	30.0	NaN
1	90.0	45.0	40.0
2	NaN	56.0	80.0
3	95.0	NaN	98.0

	First Score	Second Score	Third Score
0	100.0	30.0	-99.0
1	90.0	45.0	40.0
2	-99.0	56.0	80.0
3	95.0	-99.0	98.0

Using interpolate() function to fill the missing values

```
# importing pandas as pd
import pandas as pd

# Creating the dataframe
df = pd.DataFrame( {"A":[12, 4, 5, None, 1],
                    "B":[None, 2, 54, 3, None],
                    "C":[20, 16, None, 3, 8],
                    "D":[14, 3, None, None, 6] } )

print(df.interpolate(method = 'linear', limit_direction = 'forward'))
```


	A	B	C	D
0	12.0	NaN	20.0	14.0
1	4.0	2.0	16.0	3.0
2	5.0	54.0	NaN	NaN
3	NaN	3.0	3.0	NaN
4	1.0	NaN	8.0	6.0

	A	B	C	D
0	12.0	NaN	20.0	14.0
1	4.0	2.0	16.0	3.0
2	5.0	54.0	9.5	4.0
3	3.0	3.0	3.0	5.0
4	1.0	3.0	8.0	6.0

As we can see the output, values in the first row could not get filled as the direction of filling of values is forward and there is no previous value which could have been used in interpolation.

Dropping missing values using dropna()

Dropping rows with at least 1 null value.

```
import pandas as pd
import numpy as np
```

```
# dictionary of lists
```

```
dict = {'First Score':[100, 90, np.nan, 95],
        'Second Score': [30, np.nan, 45, 56],
        'Third Score':[52, 40, 80, 98],
        'Fourth Score':[np.nan, np.nan, np.nan, 65]}
```

```
# creating a dataframe from dictionary
```

```
df = pd.DataFrame(dict)
```

```
print(df)
```

```
# using dropna() function
```

```
print(df.dropna())
```

	First Score	Second Score	Third Score	Fourth Score
0	100.0	30.0	52	NaN
1	90.0	NaN	40	NaN
2	NaN	45.0	80	NaN
3	95.0	56.0	98	65.0

	First Score	Second Score	Third Score	Fourth Score
3	95.0	56.0	98	65.0

Drop rows whose all data is missing

```
# importing pandas as pd
import pandas as pd
```

```
# importing numpy as np
import numpy as np
```

```
# dictionary of lists
dict = {'First Score':[100, np.nan, np.nan, 95],
        'Second Score': [30, np.nan, 45, 56],
        'Third Score':[52, np.nan, 80, 98],
        'Fourth Score':[np.nan, np.nan, np.nan, 65]}
```

```
df = pd.DataFrame(dict)
```

```
# using dropna() function
df.dropna(how = 'all')
```

	First Score	Second Score	Third Score	Fourth Score
0	100.0	30.0	52.0	NaN
2	NaN	45.0	80.0	NaN
3	95.0	56.0	98.0	65.0

Drop a columns which have at least 1 missing values

```
# importing pandas as pd
import pandas as pd
```

```
# importing numpy as np
import numpy as np

# dictionary of lists
dict = {'First Score':[100, np.nan, np.nan, 95],
        'Second Score': [30, np.nan, 45, 56],
        'Third Score':[52, np.nan, 80, 98],
        'Fourth Score':[60, 67, 68, 65]}

# creating a dataframe from dictionary
df = pd.DataFrame(dict)

# using dropna() function
df.dropna(axis = 1)
```

	First Score	Second Score	Third Score	Fourth Score
0	100.0	30.0	52.0	60
1	NaN	NaN	NaN	67
2	NaN	45.0	80.0	68
3	95.0	56.0	98.0	65

	Fourth Score
0	60
1	67
2	68
3	65

Iterating over rows and columns

Iteration is a general term for taking each item of something, one after another. Pandas DataFrame consists of rows and columns so, in order to iterate over dataframe, we have to iterate a dataframe like a dictionary.

In Pandas Dataframe we can iterate an element in two ways:

- ✓ Iterating over rows
- ✓ Iterating over columns

Iterating over rows :

In order to iterate over rows, we can use three function `iteritems()`, `iterrows()`, `itertuples()`. These three function will help in iteration over rows.

```
import pandas as pd
dict = { 'name':["aparna", "pankaj", "sudhir", "Geeku"],
```

```

    'degree': ["MBA", "BCA", "M.Tech", "MBA"],
    'score': [90, 40, 80, 98] }
df = pd.DataFrame(dict)
for idx, val in df.iterrows( ):
    print(idx, "--- ", val)
    print( )

```

```

0 --- name   aparna
degree   MBA
score    90
Name: 0, dtype: object
1 --- name   pankaj
degree   BCA
score    40
Name: 1, dtype: object
2 --- name   sudhir
degree   M.Tech
score    80
Name: 2, dtype: object
3 --- name   Geeku
degree   MBA
score    98
Name: 3, dtype: object

```

Iteration over rows using iteritems()

In order to iterate over rows, we use iteritems() function this function iterates over each column as key, value pair with label as key and column value as a Series object.

```
import pandas as pd
```

```
dict = {'name': ["aparna", "pankaj", "sudhir", "Geeku"],
```

```
'degree': ["MBA", "BCA", "M.Tech", "MBA"],  
'score': [90, 40, 80, 98]}
```

```
df = pd.DataFrame(dict)
```

```
for key, value in df.iteritems():  
    print("Key-",key)  
    print(value)  
    print()
```

Key- name

0 aparna

1 pankaj

2 sudhir

3 Geeku

Name: name, dtype: object

Key- degree

0 MBA

1 BCA

2 M.Tech

3 MBA

Name: degree, dtype: object

Key- score

0 90

1 40

2 80

3 98

Name: score, dtype: int64

Iteration over rows using itertuples()

In order to iterate over rows, we apply a function `itertuples()` this function return a tuple for each row in the DataFrame. The first element of the tuple will be the row's corresponding index value, while the remaining values are the row values.

```
import pandas as pd
```

```
dict = {'name': ["aparna", "pankaj", "sudhir", "Geeku"],  
        'degree': ["MBA", "BCA", "M.Tech", "MBA"],  
        'score': [90, 40, 80, 98]}
```

```
df = pd.DataFrame(dict)
```

```
for i in df.itertuples():  
    print(i)
```

```
Pandas(Index=0, name='aparna', degree='MBA', score=90)  
Pandas(Index=1, name='pankaj', degree='BCA', score=40)  
Pandas(Index=2, name='sudhir', degree='M.Tech', score=80)  
Pandas(Index=3, name='Geeku', degree='MBA', score=98)
```

Working With Text Data

- Series and Indexes are equipped with a set of string processing methods that make it easy to operate on each element of the array.
- Perhaps most importantly, these methods exclude missing/NA values automatically.
- These are accessed via the str attribute and generally, have names matching the equivalent (scalar) built-in string methods.

Lowercasing and Uppercasing a Data

1. `str.lower()` this function converts all uppercase characters to lowercase.
2. `str.upper()` this function converts all lowercase characters to uppercase.

```
import pandas as pd
```

```
# Define a dictionary containing employee data  
data = {'Name':['Jai', 'Princi', 'Gaurav', 'Anuj'],  
        'Age':[27, 24, 22, 32],  
        'Address':['Delhi', 'Kanpur', 'Allahabad', 'Kannauj'],  
        'Qualification':['Msc', 'MA', 'MCA', 'Phd']}
```

```
# Convert the dictionary into DataFrame  
df = pd.DataFrame(data)  
# converting and overwriting values in column  
df["Name"] = df["Name"].str.lower()  
print(df)
```

Splitting and Replacing a Data

- In order to split a data, we use `str.split()`
- This function returns a list of strings after breaking the given string by the specified separator
- It can only be applied to an individual string.
- Pandas `str.split()` method can be applied to a whole series.
- `.str` has to be prefixed every time before calling this method to differentiate it from the Python's default function otherwise, it will throw an error.
- In order to replace a data, we use `str.replace()` this function works like Python `.replace()` method only, but it works on Series too.
- Before calling `.replace()` on a Pandas series, `.str` has to be prefixed in order to differentiate it from the Python's default replace method.

Splitting and Replacing a Data

```
import pandas as pd
```

```
data = {'Name':['Jai', 'Princi', 'Gaurav', 'Anuj'],
        'Age':[27, 24, 22, 32],
        'Address':['Nagpur', 'Kanpur', 'Allahabad', 'Knnuaj'],
        'Qualification':['Msc', 'MA', 'MCA', 'Phd']}
```

```
df = pd.DataFrame(data)
```

```
df.dropna(inplace = True)
```

```
df["Address"] = df["Address"].str.split("a", expand = True)
```

```
print(df)
```

	Name	Age	Address	Qualification
0	Jai	27	Nagpur	Msc
1	Princi	24	Kanpur	MA
2	Gaurav	22	Allahabad	MCA
3	Anuj	32	Knnuaj	Phd

	Name	Age	Address	Qualification
0	Jai	27	N	Msc
1	Princi	24	K	MA
2	Gaurav	22	All	MCA
3	Anuj	32	Knnu	Phd

Splitting and Replacing a Data

Replace

```
import pandas as pd
data = {'Name':['Jai', 'Princi', 'Gaurav', 'Anuj'],
        'Age':[27, 25, 22, 25],
        'Address':['Nagpur', 'Kanpur', 'Allahabad', 'Knnuaj'],
        'Qualification':['Msc', 'MA', 'MCA', 'Phd']}

df = pd.DataFrame(data)
print(df)
df["Age"] = df["Age"].replace( 25 , "Twenty five")
print(df)
```

	Name	Age	Address	Qualification
0	Jai	27	Nagpur	Msc
1	Princi	25	Kanpur	MA
2	Gaurav	22	Allahabad	MCA
3	Anuj	25	Knnuaj	Phd

	Name	Age	Address	Qualification
0	Jai	27	Nagpur	Msc
1	Princi	Twenty five	Kanpur	MA
2	Gaurav	22	Allahabad	MCA
3	Anuj	Twenty five	Knnuaj	Phd

Concatenation of Data

`str.cat()` this function is used to concatenate strings to the passed caller series of string.

```
import pandas as pd
data = {'Name':['Jai', 'Princi', 'Gaurav', 'Anuj'],
        'Age':[27, 24, 22, 32],
        'Address':['Nagpur', 'Kanpur', 'Allahabad', 'Kannuaj'],
        'Qualification':['Msc', 'MA', 'MCA', 'Phd']}

df = pd.DataFrame(data)

new = df["Address"].copy()

df["Name"] = df["Name"].str.cat(new, sep = ", ")

print(df)
```


	Name	Age	Address	Qualification
0	Jai, Nagpur	27	Nagpur	Msc
1	Princi, Kanpur	24	Kanpur	MA
2	Gaurav, Allahabad	22	Allahabad	MCA
3	Anuj, Kannuaj	32	Kannuaj	Phd

Removing Whitespaces of Data

- In order to remove a whitespaces, we use `str.strip()`, `str.rstrip()`, `str.lstrip()` these function used to handle white spaces(including New line) in any text data.
- As it can be seen in the name, `str.lstrip()` is used to remove spaces from the left side of string, `str.rstrip()` to remove spaces from right side of the string and `str.strip()` removes spaces from both sides.
- Since these are pandas function with same name as Python's default functions, `.str` has to be prefixed to tell the compiler that a Pandas function is being called.

```
import pandas as pd
```

```
data = {'Name':['Jai', 'Princi', 'Gaurav', 'Anuj'],
        'Age':[27, 24, 22, 32],
        'Address':['Nagpur junction', 'Kanpur junction',
                   'Nagpur junction', 'Kannuaj junction'],
        'Qualification':['Msc', 'MA', 'MCA', 'Phd']}
```

```
df = pd.DataFrame(data)
```

```
new = df["Address"].replace("Nagpur junction", " Nagpur junction ").copy()
```

```
print(new.str.strip()==" Nagpur junction")
print(new.str.strip()=="Nagpur junction ")
print(new.str.strip()==" Nagpur junction ")
```

Pandas str methods:

Function	Description
<code>str.lower()</code>	Method to convert a string's characters to lowercase
<code>str.upper()</code>	Method to convert a string's characters to uppercase
<code>str.find()</code>	Method is used to search a substring in each string present in a series

<code>str.rfind()</code>	Method is used to search a substring in each string present in a series from the Right side
<code>str.findall()</code>	Method is also used to find substrings or separators in each string in a series
<code>str.isalpha()</code>	Method is used to check if all characters in each string in series are alphabetic(a-z/A-Z)
<code>str.isdecimal()</code>	Method is used to check whether all characters in a string are decimal
<code>str.title()</code>	Method to capitalize the first letter of every word in a string
<code>str.len()</code>	Method returns a count of the number of characters in a string
<code>str.replace()</code>	Method replaces a substring within a string with another value that the user provides
<code>str.contains()</code>	Method tests if pattern or regex is contained within a string of a Series or Index
<code>str.extract()</code>	Extract groups from the first match of regular expression pattern.
<code>str.startswith()</code>	Method tests if the start of each string element matches a pattern
<code>str.endswith()</code>	Method tests if the end of each string element matches a pattern
<code>str.isdigit()</code>	Method is used to check if all characters in each string in series are digits
<code>str.lstrip()</code>	Method removes whitespace from the left side (beginning) of a string
<code>str.rstrip()</code>	Method removes whitespace from the right side (end) of a string
<code>str.strip()</code>	Method to remove leading and trailing whitespace from string
<code>str.split()</code>	Method splits a string value, based on an occurrence of a user-specified value
<code>str.join()</code>	Method is used to join all elements in list present in a series with passed delimiter
<code>str.cat()</code>	Method is used to concatenate strings to the passed caller series of string.
<code>str.repeat()</code>	Method is used to repeat string values in the same position of passed series itself
<code>str.get()</code>	Method is used to get element at the passed position

<code>str.partition()</code>	Method splits the string only at the first occurrence unlike <code>str.split()</code>
<code>str.rpartition()</code>	Method splits string only once and that too reversely. It works in a similar way like <code>str.partition()</code> and <code>str.split()</code>
<code>str.pad()</code>	Method to add padding (whitespaces or other characters) to every string element in a series
<code>str.swapcase()</code>	Method to swap case of each string in a series

Day-15:

Q.1 Multiple Choice Question

- Which function is not used for removing missing data in Pandas Module?
 - fillna().
 - replace().
 - interpolate().
 - isnull().
- Which function is used for detect missing data in Pandas Module?
 - isnull().
 - replace().
 - dropna().
 - fillna().
- Which one is not true iterating rows from dataframe from Pandas?
 - iteritem() – access key-value pair of items.
 - iterrows() – access key-row pair.
 - itertuples() - access rows as tuple.
 - itercolumns() - access key-columns pair.
- Which one is invalid str function used for processing Dataframe?
 - str.lower().
 - str.contains().
 - str.title().
 - str.reverse().
- Which is not used to remove whitespace for Dataframe?
 - str.strip().
 - str.rstrip().
 - str.lstrip().
 - str.astrip().

Q.2 Records of T20 Matches conducted a different cities are available in a CSV file.
The File have multiple Columns

Id, Season, city , date , team1 , team2 , tos_winner, toss_decision, result, dl_applied, winner, win_by_run, win_by_wickets, Player_of_match, venue, umpire1, umpire2, umpire3.

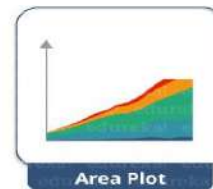
By using Pandas , find the following

- List of cities(unique values only) where matches conducted.
- List of team(unique values only) participated in the match.
- Count of matches conducted in year -2010, 2015, 2017
- Count of matches conducted at 'Bangalore' city.
- Count of matches conducted in April-2010.
- Count of matched in which result is 'TIE'
- Count of matches in which 'SK Raina' was the Player of the match.
- Count of tos won by each team.
- Count of player_of_macth given to 'Yuvraj Singh'.
- Name of team won the maximum match in Season-2014 .

Python - Matplotlib

- matplotlib.pyplot is a plotting library used for 2D graphics in python programming language. It can be used in python scripts, shell, web application servers and other graphical user interface toolkits.
- There are several toolkits which are available that extend python matplotlib functionality
 - Basemap
 - Cartopy
 - Excel tools
 - Mplot3d
 - Natgrid

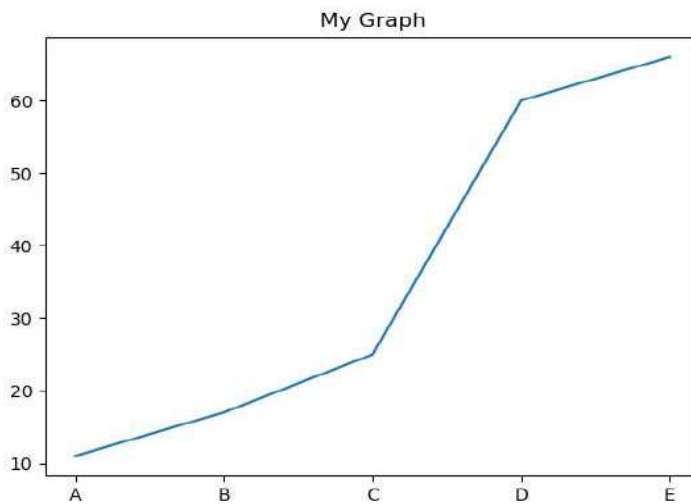
- **Types of Plots**



- **Installing Matplotlib**
 - Now, open a *cmd* window like before. Use the next set of commands to install *Matplotlib*:
 - `python -m pip install matplotlib`
- **Importing Matplotlib**
 - The package is imported into the Python script by adding the following statement
—
 - `from matplotlib import pyplot as plt`

Matplotlib Example

```
import matplotlib.pyplot as plt
#the x and y coordinates
x = [ 'A','B','C','D', 'E']
y = [ 11,17,25,60,66]
plt.title("My Graph")
# Plot the points using matplotlib
plt.plot(x, y)
plt.show()
```



Matplotlib Functions

Function	Description
plot()	Plot y versus x as lines and/or markers.
xlabel()	Set the label for the x-axis.
ylabel()	Set the label for the y-axis.
show()	Display a figure.
axis()	Convenience method to get or set some axis properties.
scatter()	A scatter plot of y vs x with varying marker size and/or color.
subplot()	Add a subplot to the current figure.
bar()	Make a bar plot.
suptitle()	Add a centered title to the figure.
close()	Close a figure window.

Formatting the style of your plot

1. For every x, y pair of arguments, there is an optional third argument which is the format string that indicates the color and line type of the plot.
2. The letters and symbols of the format string are from MATLAB, and you concatenate a color string with a line style string.
3. The default format string is 'b-', which is a solid blue line.
4. For example, to plot with red circles, we write

```
plt.plot([1, 2, 3, 4], [1, 4, 9, 16], 'ro')  
plt.axis([0, 6, 0, 20])  
plt.show()
```

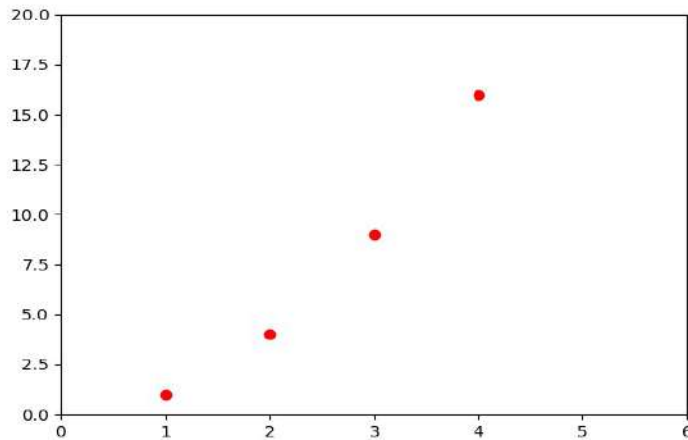
Note: The `axis()` command takes a list of `[xmin, xmax, ymin, ymax]`

Color Codes

character	color
'b'	blue
'g'	green
'r'	red
'c'	cyan
'm'	magenta
'y'	yellow
'k'	black
'w'	white

Line Style

character	description
' - '	solid line style
' - - '	dashed line style
' - . '	dash-dot line style
' : '	dotted line style
' . '	point marker
' , '	pixel marker
' o '	circle marker
' v '	triangle_down marker
' ^ '	triangle_up marker
' < '	triangle_left marker
' > '	triangle_right marker
' 1 '	tri_down marker
' 2 '	tri_up marker
' 3 '	tri_left marker
' 4 '	tri_right marker
' s '	square marker
' p '	pentagon marker
' * '	star marker
' h '	hexagon1 marker
' H '	hexagon2 marker
' + '	plus marker
' x '	x marker



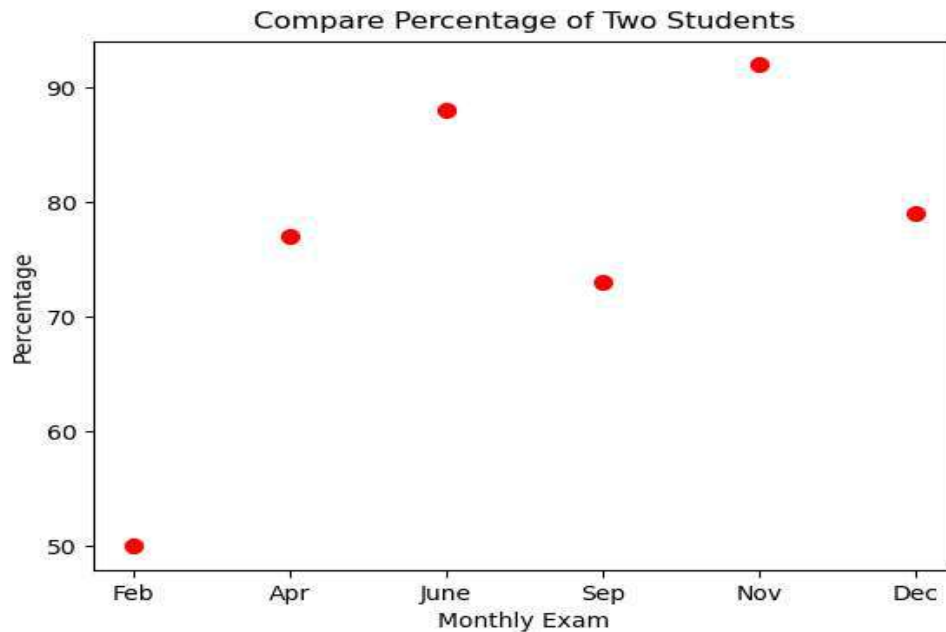
Plotting with keyword strings

There are some instances where you have data in a format that lets you access particular variables with strings. For example, with **numpy.recarray** or **pandas.DataFrame**.

Matplotlib allows you provide such an object with the **data** keyword argument. If provided, then you may generate plots with the strings corresponding to these variables.

Example

```
import pandas as pd
import matplotlib.pyplot as plt
Student1={ 'Monthly': [ 'Feb' , 'Apr' , 'June', 'Sep', 'Nov', 'Dec'],
           'Eng' : [45,67,78,58,87,89],
           'Maths': [55,87,98,88,97,69]
          }
df1=pd.DataFrame(Student1)
df1['Total']= df1['Eng']+df1['Maths']
df1['PCT']=df1['Total']/2
plt.scatter('Monthly','PCT', s=50,color='r' , data=df1)
plt.xlabel('Monthly Exam')
plt.ylabel('Percentage')
plt.title('Compare Percentage of Two Students')
plt.show( )
```

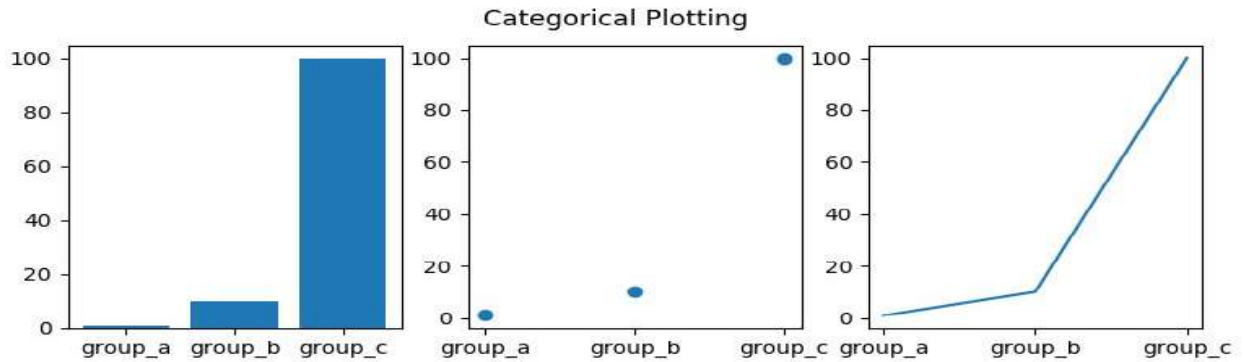


Plotting with categorical variables

It is also possible to create a plot using categorical variables. Matplotlib allows you to pass categorical variables directly to many plotting functions.

For example:

```
names = ['group_a', 'group_b', 'group_c']
values = [1, 10, 100]
plt.figure(figsize=(9, 3))
plt.subplot(131)
plt.bar(names, values)
plt.subplot(132)
plt.scatter(names, values)
plt.subplot(133)
plt.plot(names, values)
plt.suptitle('Categorical Plotting')
plt.show( )
```



Controlling line properties

Lines have many attributes that you can set: linewidth, dash style, antialiased, etc. There are several ways to set line properties.

- 1. Use **keyword args**:

```
plt.plot(x, y, linewidth=2.0)
```

- 2. Use **the setter methods** of a Line2D instance. plot returns a list of Line2D objects;

e.g., `line1 = plt.plot(x1, y1).`

Use the **setp()** command.

1. The example below uses a MATLAB-style command to set multiple properties on a list of lines.
2. **setp** works transparently with a list of objects or a single object. You can either use python keyword arguments or MATLAB-style string/value pairs:

Example

```
lines = plt.plot(x1, y1, x2, y2)
# use keyword args
plt.setp(lines, color='r', linewidth=2.0)
# or MATLAB style string value pairs
plt.setp(lines, 'color', 'r', 'linewidth', 2.0)
```

Scatter Plot

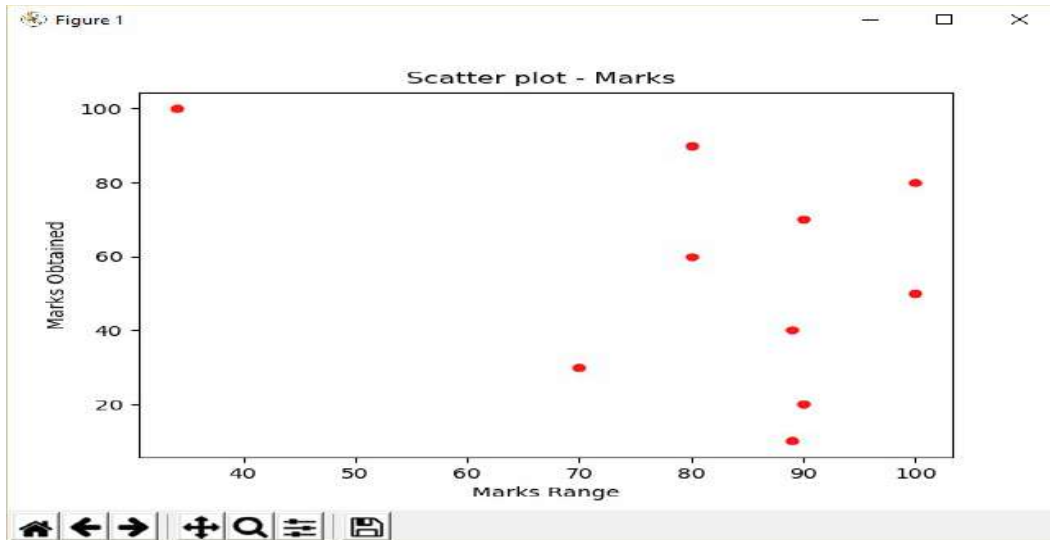
1. This types of plot shows all individual data points. Here, they aren't connected with lines.
2. Each data point has the value of the x-axis value and the value from the y-axis values.
3. This type of plot can be used to display trends or correlations.
4. In data science, it shows how 2 variables compare.
5. To make a scatter plot with Matplotlib, we can use the `plt.scatter()` function.
6. Again, the first argument is used for the data on the horizontal axis, and the second - for the vertical axis.

Example

```
import numpy as np
import matplotlib.pyplot as plt

# Create data
Marks = [89, 90, 70, 89, 100, 80, 90, 100, 80, 34]
Marks_Range=[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
colors = 'r'
circle_area = 25
contrast=0.9 # has value from 0 - 1

# Plot
plt.scatter(Marks,Marks_Range, s=circle_area, c=colors, alpha=contrast)
plt.title('Scatter plot - Marks ')
plt.xlabel('Marks Range')
plt.ylabel('Marks Obtained')
plt.show()
```



Scatter plot with groups

Data can be classified in several groups.

Example

```
import numpy as np
```

```
import matplotlib.pyplot as plt
```

Create data

```
g1 = np.array( [ [89, 90, 70, 89, 100, 80, 90, 100, 80, 34],
                 [1,2,3,4,5,6,7,8,9,10] ] )
```

```
g2 = np.array([ [30, 29, 49, 48, 100, 48, 38, 45, 20, 30],
                 [1,2,3,4,5,6,7,8,9,10] ])
```

```
g3 = np.array([ [69, 90, 70, 59, 70, 80, 40, 80, 55, 24],
                 [1,2,3,4,5,6,7,8,9,10] ])
```

```
data = (g1, g2, g3)
```

```
colors = ("red", "green", "blue")
```

```
groups = ("Red House", "Green House", "Blue House")
```

Create plot

```
fig = plt.figure()
```

```
ax = fig.add_subplot(1,1,1)
```

```
x, y = g1
```

```
ax.scatter(y, x, alpha=0.8, c='red', edgecolors='none', s=30, label='Red House')
```

```
x, y = g2
```

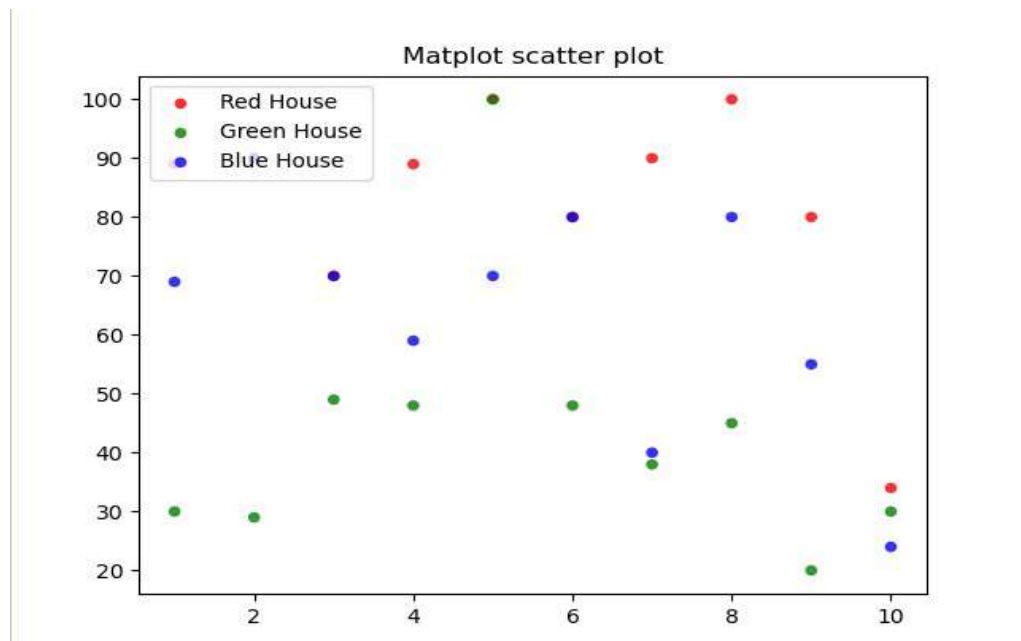
```
ax.scatter(y, x, alpha=0.8, c='green', edgecolors='none', s=30, label='Green House')
```

```
x, y = g3
```

```
ax.scatter(y, x, alpha=0.8, c='blue', edgecolors='none', s=30, label='Blue House')
```

```
#for data, color, group in zip(data, colors, groups):
# x, y = data
# ax.scatter(y, x, alpha=0.8, c=color, edgecolors='none', s=30, label=group)

plt.title('Matplot scatter plot')
plt.legend(loc=2)
plt.show()
```



Bar Chart

Bar chart represents categorical data with rectangular bars. Each bar has a height corresponds to the value it represents.

Syntax `plt.bar()`

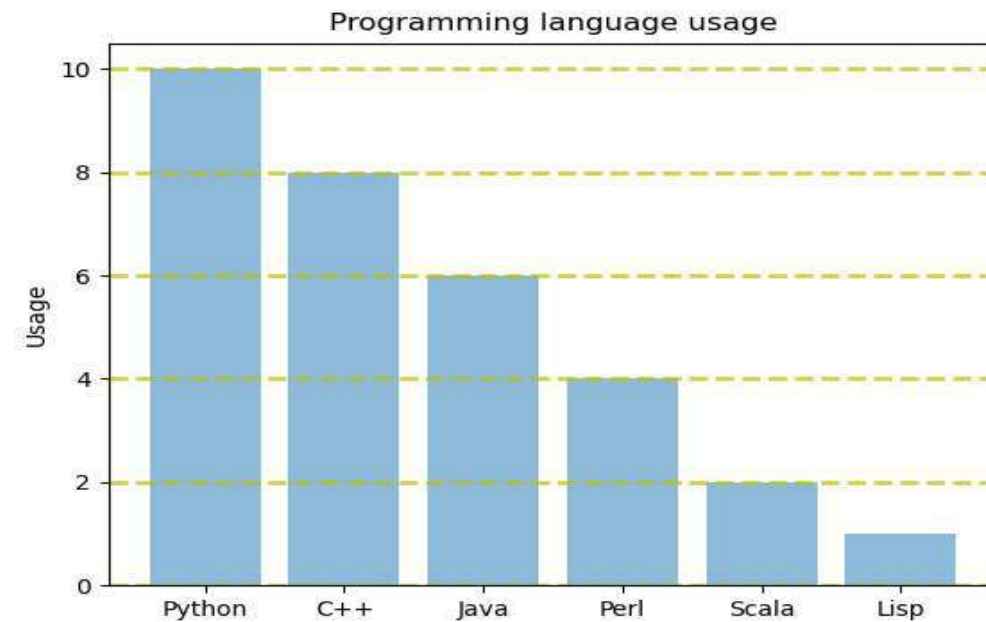
Example

```
import numpy as np
import matplotlib.pyplot as plt

objects = ('Python', 'C++', 'Java', 'Perl', 'Scala', 'Lisp')
y_pos = np.arange(len(objects))
performance = [10,8,6,4,2,1]

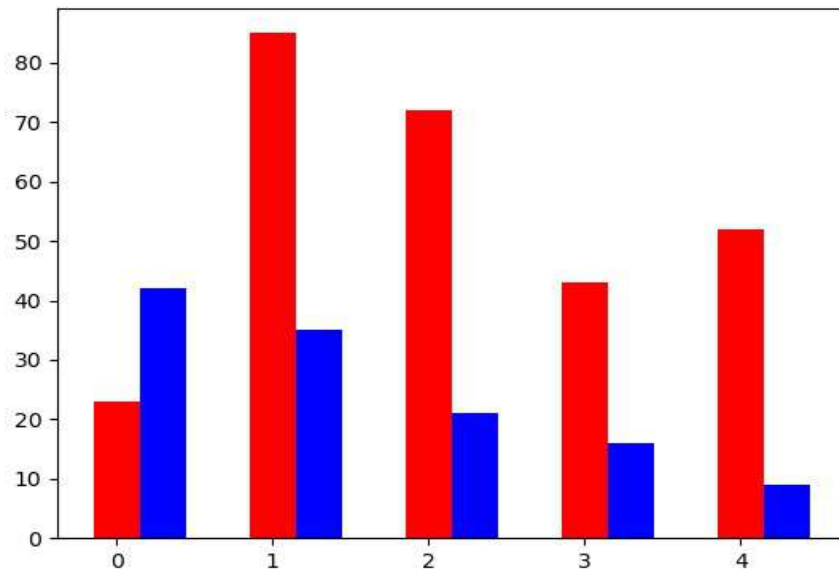
plt.grid(color='y', linestyle='--', linewidth=2, axis='y', alpha=0.7)
plt.bar(y_pos, performance, align='center', alpha=0.5)
plt.xticks(y_pos, objects)
```

```
plt.ylabel('Usage')
plt.title('Programming language usage')
plt.show()
```



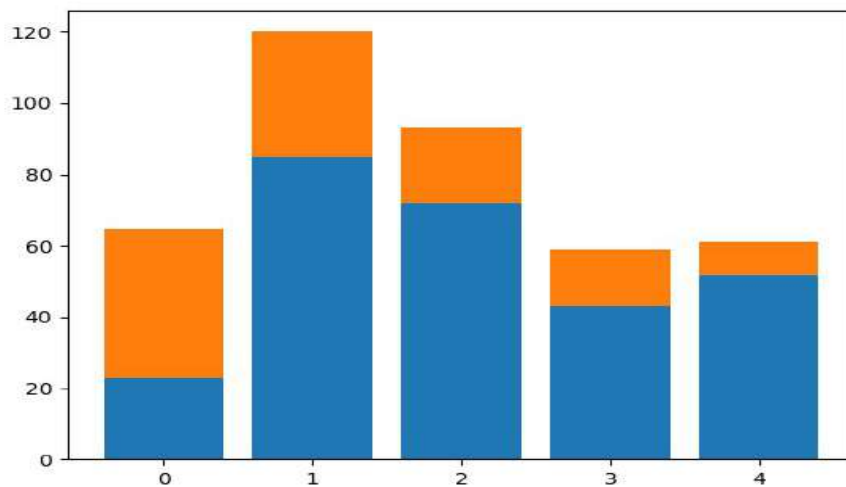
Multiple Bar Chart

```
import numpy as np
import matplotlib.pyplot as plt
data1 = [23, 85, 72, 43, 52]
data2 = [42, 35, 21, 16, 9]
width = 0.3
plt.bar(np.arange(len(data1)), data1, color='r', width=width)
plt.bar(np.arange(len(data2)) + width, data2, color='b', width=width)
plt.show()
```



Stack Bar Chart

```
import numpy as np
import matplotlib.pyplot as plt
data1 = [23, 85, 72, 43, 52]
data2 = [42, 35, 21, 16, 9]
plt.bar(range(len(data1)), data1)
plt.bar(range(len(data2)), data2, bottom=data1)
plt.show()
```



Compare the two data Series

You can compare two data series using this Matplotlib code:

```
import numpy as np
import matplotlib.pyplot as plt

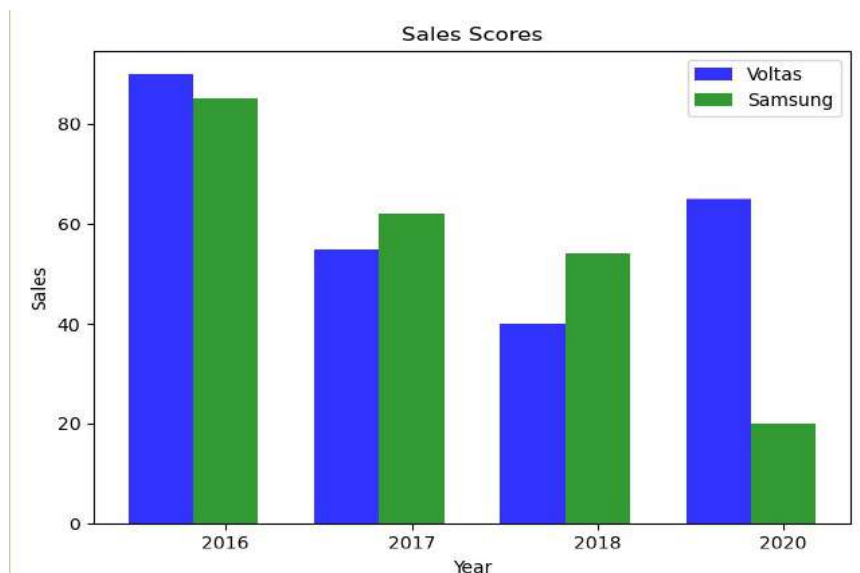
# data to plot
n_groups = 4
Sales_Voltas = (90, 55, 40, 65)
Sales_Samsung = (85, 62, 54, 20)

# create plot
y_pos = np.arange(n_groups)
bar_width = 0.35
opacity = 0.8

rects1 = plt.bar(y_pos, Sales_Voltas, bar_width, alpha=opacity, color='b', label='Voltas')
rects2 = plt.bar(y_pos + bar_width, Sales_Samsung, bar_width, alpha=opacity, color='g',
label='Samsung')

plt.xlabel('Year')
plt.ylabel('Sales')
plt.title('Sales Scores')
plt.xticks(y_pos + bar_width, Year_Sales)
plt.legend()

plt.tight_layout()
plt.show()
```

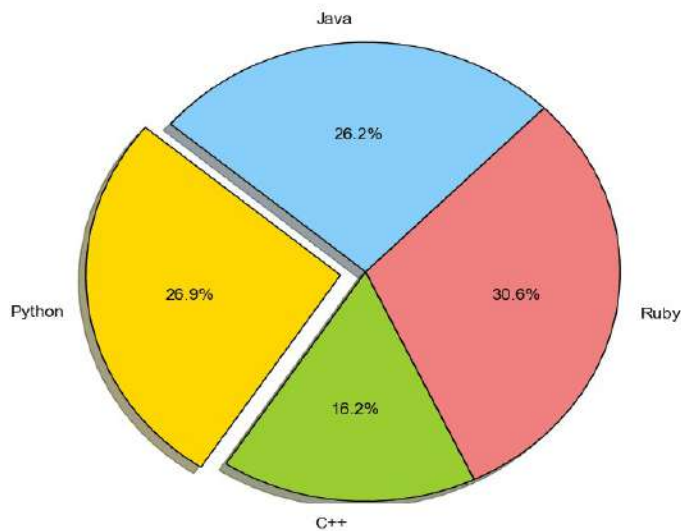


Pie chart

Pie chart: a circular plot, divided into slices to show numerical proportion. They are widely used in the business world.

Example

```
import matplotlib.pyplot as plt
labels = 'Python', 'C++', 'Ruby', 'Java'
sizes = [215, 130, 245, 210]
colors = ['gold', 'yellowgreen', 'lightcoral', 'lightskyblue']
explode = (0.1, 0, 0, 0) # explode 1st slice
plt.pie(sizes, explode=explode, labels=labels, colors=colors,
autopct='%1.1f%%', shadow=True, startangle=140)
plt.axis('equal')
plt.show()
```



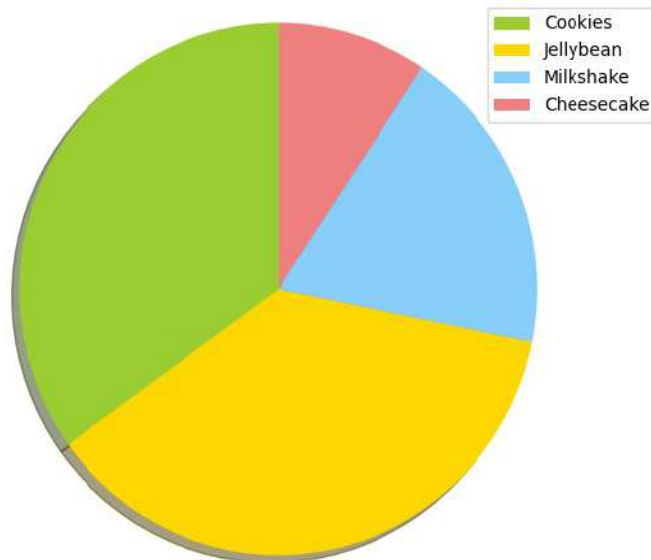
To add a legend use the **plt.legend()** function:

Legend Location

- best
- upper right
- upper left
- lower left
- lower right
- right
- center left
- center right

- lower center
- upper center
- center

```
import matplotlib.pyplot as plt
labels = ['Cookies', 'Jellybean', 'Milkshake', 'Cheesecake']
sizes = [38.4, 40.6, 20.7, 10.3]
colors = ['yellowgreen', 'gold', 'lightskyblue', 'lightcoral']
plt.pie(sizes, colors=colors, shadow=True, startangle=90, labels=labels)
plt.legend( loc="best")
plt.axis('equal')
plt.tight_layout()
plt.show()
```



Day-16

Q.1. Multiple Choice Question

1. Which one is not a valid graph type in Matplotlib?
 - a. Scatter.
 - b. Pie.
 - c. Bar.
 - d. Platter.
2. Which one is not valid function in Pyplot?
 - a. `axis()` get or set axis.
 - b. `title()` add title for graph.
 - c. `subtitle()` add the sub title for graph.
 - d. `show()` show the graph.
3. Which one is not for related for label in Graph?
 - a. `xlabel()` - set the label for X axis.
 - b. `ylabel()` - set the label for Y axis.
 - c. `subtitle()` - set the title for graph.
 - d. `legend()` - to show the legend for graph.
4. Which one is not true statement in Pyplot?
 - a. `bar()` - plot the bar graph based on x and y.
 - b. `scatter()` - plot the scatter graph based on x and y.
 - c. `show()` - show the graph on screen.
 - d. `line()` - plot the line graph based on x and y.
5. Which is not valid for graph in Pyplot?
 - a. `subplot()` - used to plot multiple graph on canvas.
 - b. `subplot(131)` - used to plot the 3 sub graph on same canvas.
 - c. `axis()` - used to specify xmin, xmax, ymin, ymax for axis.
 - d. `setlinep()` - used to set properties of lines.

Q. 2 Data of COVID cases in various states in the month of May-2020 and, June-2020.

State	May-2020	June-2020
Maharashtra	28078	39678
Rajasthan	17067	13456
Uttar Pradesh	12670	19654
Kerala	19765	10879
Panjab	18566	12009
Jharkhand	5700	9100
Haryana	3450	8700
Orisha	2300	7800

1. Draw bar chart to compare the COVID cases in various states in the month of May-2020 and, June-2020.
2. Draw a Pie Chart to represent the share of states in positive cases in the Month of May-2020. Show the legends and explode the state with highest share.
3. Draw a Line Chart to compare the cases of all the states in May-2020 and June-2020.
4. Draw a Line , Pie and Bar Chart on a single graph using subplot to compare the cases of states in the month of June-2020.
5. Draw a Pie Chart showing the share of cases of top 4 states and cases of all remaining states as other.

GUI Programming

- GUI stands for graphical user interface, a GUI is an interface that uses icons or other visual indicators to interact with electronic devices, rather than only text via a command line.
- For example, all versions of Microsoft Windows is a GUI, whereas MS-DOS is a command line.

Tkinter

1. Tk was developed as a GUI extension for the Tcl scripting language by John Ousterhout.
2. The first release was in 1991. Tk proved as extremely successful in the 1990's, because it is easier to learn and to use than other toolkits.
3. Tkinter is an inbuilt Python module used to create simple GUI apps.
4. It is the most commonly used module for GUI apps in the Python.
5. The name Tkinter comes from Tk interface.

Opening New Window

First GUI Program

import tkinter

```
window = tkinter.Tk()
window.title("GUI") # to rename the title of the window
# pack is used to show the object in the window
label = tkinter.Label(window, text = "Hello World!").pack( )
window.mainloop( )
```



- import the module **tkinter**.
- Initialize the window manager with the **tkinter.Tk()** method and assign it to a variable **window**. This method creates a blank window with close, maximize and minimize buttons.

- Rename the title of the window as you like with the **window.title(title_of_the_window)**.
- **Label** is used to insert some objects into the **window**. Here, we are adding a **Label** with some text.
- **pack()** attribute of the widget is used to display the **widget** in a size it requires.
- Finally, the **mainloop()** method to display the **window** until you manually close it.

Placing Widgets or Component in Window

1. Tkinter allows to access the geometric configuration of the widgets which can organize the widgets in the parent windows.
2. There are mainly three geometry manager classes class.
 - **pack() method:** It organizes the widgets in blocks before placing in the parent widget.
 - **grid() method:** It organizes the widgets in grid (table-like structure) before placing in the parent widget.
 - **place() method:** It organizes the widgets by placing them on specific positions directed by the programmer.

Tkinter Widgets

1. **Button** - The Button widget is used to display buttons in your application.
2. **Canvas** - The Canvas widget is used to draw shapes, such as lines, ovals, polygons and rectangles, in your application.
3. **Checkbutton** - The Checkbutton widget is used to display a number of options as checkboxes. The user can select multiple options at a time.
4. **Entry** - The Entry widget is used to display a single-line text field for accepting values from a user.
5. **Frame** - The Frame widget is used as a container widget to organize other widgets.
6. **Label** - The Label widget is used to provide a single-line caption for other widgets. It can also contain images.
7. **Listbox** - The Listbox widget is used to provide a list of options to a user.
8. **Menubutton** - The Menubutton widget is used to display menus in your application.
9. **Menu** - The Menu widget is used to provide various commands to a user. These commands are contained inside Menubutton.
10. **Message** - The Message widget is used to display multiline text fields for accepting values from a user.
11. **Radiobutton** - The Radiobutton widget is used to display a number of options as radio buttons. The user can select only one option at a time.
12. **Scale** - The Scale widget is used to provide a slider widget.
13. **Scrollbar** - The Scrollbar widget is used to add scrolling capability to various widgets, such as list boxes.
14. **Text** - The Text widget is used to display text in multiple lines.
15. **Toplevel** - The Toplevel widget is used to provide a separate window container.

16. **Spinbox** -The Spinbox widget is a variant of the standard Tkinter Entry widget, which can be used to select from a fixed number of values.
17. **PanedWindow** -A PanedWindow is a container widget that may contain any number of panes, arranged horizontally or vertically.
18. **LabelFrame** -A labelframe is a simple container widget. Its primary purpose is to act as a spacer or container for complex window layouts.
19. **tkMessageBox** -This module is used to display message boxes in your applications.

Standard attributes

Common attributes of the widgets such as sizes, colors and fonts are specified.

- Dimensions
- Colors
- Fonts
- Anchors
- Relief styles
- Bitmaps
- Cursors

Label

It refers to the display box where you can put any text or image which can be updated any time as per the code.

Syntax :

w=Label(master, option=value)

master is the parameter used to represent the parent window.

- **bg**: to set the normal background color.
- **command**: to call a function.
- **font**: to set the font on the button label.
- **image**: to set the image on the button.
- **width**: to set the width of the button.
- **height**” to set the height of the button.

Example

```
from tkinter import *
root = Tk()
w = Label(root, text='This is Label')
w.pack()
root.mainloop()
```




Button

To add a button in your application, this widget is used.

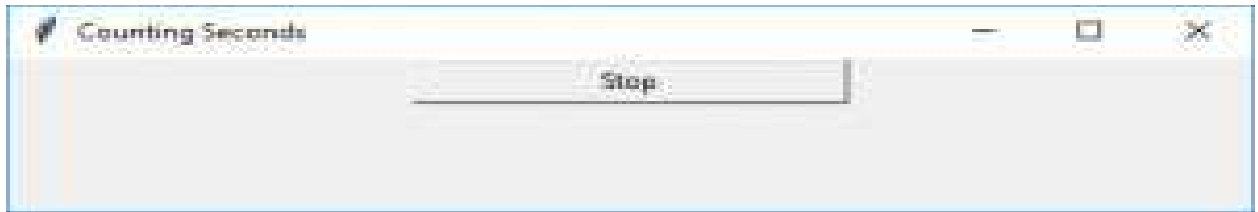
The general syntax is:

```
w=Button(master, option=value)
```

- **activebackground:** to set the background color when button is under the cursor.
- **activeforeground:** to set the foreground color when button is under the cursor.
- **bg:** to set the normal background color.
- **command:** to call a function.
- **font:** to set the font on the button label.
- **image:** to set the image on the button.
- **width:** to set the width of the button.
- **height:** to set the height of the button.

Example-2 for Button

```
import tkinter as tk
r = tk.Tk()
r.title('Counting Seconds')
button = tk.Button(r, text='Stop', width=25, command=r.destroy)
button.pack()
r.mainloop()
```



Example-2 for Button

```
import tkinter as tk
def show():
    tk.Message(r,text="Hello", fg='red', font='Arial', bg='yellow', relief=tk.RAISED).pack()
    return
r = tk.Tk()
r.title('Show Message')
button = tk.Button(r, text='Click Here', font='Arial', fg='brown', width=25, command=show)
button.pack()
r.mainloop()
```



Entry

It is used to input the single line text entry from the user.. For multi-line text input, Text widget is used.

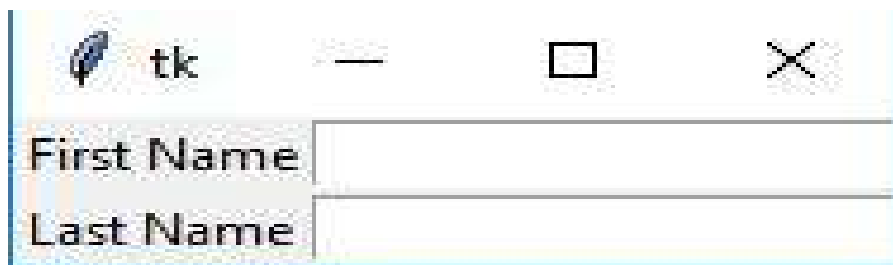
Syntax

w=Entry(master, option=value)

- **bd:** to set the border width in pixels.
- **bg:** to set the normal background color.
- **cursor:** to set the cursor used.
- **command:** to call a function.
- **highlightcolor:** to set the color shown in the focus highlight.
- **width:** to set the width of the button.
- **height:** to set the height of the button.

Example

```
from tkinter import *
master = Tk()
Label(master, text='First Name').grid(row=0)
Label(master, text='Last Name').grid(row=1)
e1 = Entry(master)
e2 = Entry(master)
e1.grid(row=0, column=1)
e2.grid(row=1, column=1)
mainloop()
```



Exempl-2 for Entry

```
from tkinter import *
def show():
    age=(vAge.get())
    if age>=18: msg= "Hello "+ vName.get() + ", you are eligible to Vote !"
    else:  msg= "Hello "+ vName.get() + ", you are not eligible to Vote !"
    m1=Message(master,text=msg, fg='red', bg='yellow', width=100,relief=RAISED)
    m1.grid(row=3 , columnspan=2)
    return
master = Tk()
Label(master, text='First Name').grid(row=0)
Label(master, text='Age').grid(row=1)
vName=StringVar()
vAge=IntVar()
e1=Entry(master, textvariable=vName).grid(row=0, column=1)
```

```
e2=Entry(master, textvariable=vAge).grid(row=1, column=1)
b1 = Button(master, text='Click Here', font='Arial', fg='brown', width=25, command=show)
b1.grid(row=2,columnspan=2)
mainloop()
```



CheckBox

To select any number of options by displaying a number of options to a user as toggle buttons.

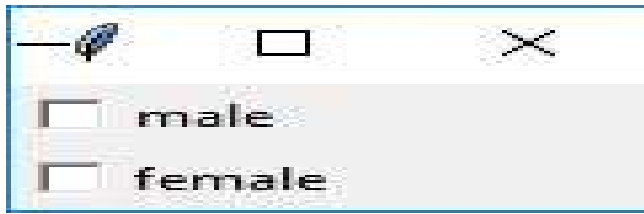
Syntax :

w = CheckButton(master, option=value)

- **Title:** To set the title of the widget.
- **activebackground:** to set the background color when widget is under the cursor.
- **activeforeground:** to set the foreground color when widget is under the cursor.
- **bg:** to set the normal background color.
- **command:** to call a function.
- **font:** to set the font on the button label.
- **image:** to set the image on the widget.

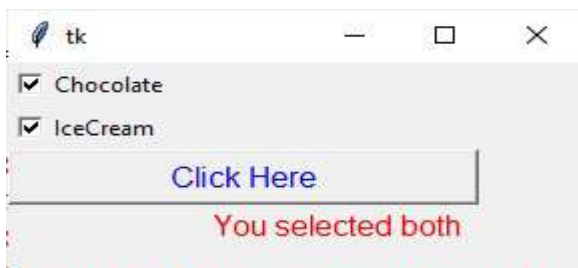
Example-1 for Check Button

```
from tkinter import *
master = Tk()
var1 = IntVar()
Checkbutton(master, text='male', variable=var1).grid(row=0, sticky=W)
var2 = IntVar()
Checkbutton(master, text='female', variable=var2).grid(row=1, sticky=W)
mainloop()
```



Example-2 for Check Button

```
from tkinter import *
def show():
    v1=int(var1.get())
    v2=int(var2.get())
    if v1==1 and v2==1 : msg.set( "You selected both")
    if v1==1 and v2==0: msg.set( "You selected Chocolate")
    if v1==0 and v2==1: msg.set( "You selected IceCream")
    if v1==0 and v2==0: msg.set( "You selected None")
    return
master = Tk()
var1 = IntVar()
var2 = IntVar()
msg=StringVar()
Checkbutton(master, text='Chocolate', variable=var1).grid(row=1, sticky=W)
Checkbutton(master, text='IceCream', variable=var2).grid(row=2, sticky=W)
b1 = Button(master, text='Click Here', font='Arial', fg='blue', width=25, command=show)
b1.grid(row=3,columnspan=2)
Label(master, fg='red', font='Arial', textvariable=msg).grid(row=4 , column=1)
mainloop()
```



RadioButton

It is used to offer multi-choice option to the user. It offers several options to the user and the user has to choose one option.

The general syntax is:

w = RadioButton(master, option=value)

- **activebackground**: to set the background color when widget is under the cursor.
- **bg**: to set the normal background color.
- **command**: to call a function.
- **font**: to set the font on the button label.
- **image**: to set the image on the widget.
- **width**: to set the width of the label in characters.
- **height**: to set the height of the label in characters.

Example

```
from tkinter import *
root = Tk()
v = IntVar()
Radiobutton(root, text='Male', variable=v, value=1).pack(anchor=W)
Radiobutton(root, text='Female', variable=v, value=2).pack(anchor=W)
mainloop()
```



Example-2 for Radio Button

```
from tkinter import *
def show():
    v1=int(var1.get())
    if v1==1 : msg.set( "You selected Chocolate")
    if v1==2: msg.set( "You selected IceCream")
    return
master = Tk()
var1 = IntVar()
msg=StringVar()
var1.set(2)      # default selection
Radiobutton(master, text='Chocolate', variable=var1 , value=1).grid(row=1, sticky=W)
Radiobutton(master, text='IceCream', variable=var1, value=2).grid(row=2, sticky=W)
b1 = Button(master, text='Click Here', font='Arial', fg='blue', width=25, command=show)
b1.grid(row=3,columnspan=2)
Label(master, fg='red', font='Arial', textvariable=msg).grid(row=4 , column=1)
mainloop()
```



Frame

It acts as a container to hold the widgets. It is used for grouping and organizing the widgets.

Syntax :

w = Frame(master, option=value)

- master is the parameter used to represent the parent window.
- **highlightcolor**: To set the color of the focus highlight when widget has to be focused.
- **bd**: to set the border width in pixels.
- **bg**: to set the normal background color.
- **cursor**: to set the cursor used.
- **width**: to set the width of the widget.
- **height**: to set the height of the widget.

Example

```
from tkinter import *
root = Tk()
frame = Frame(root)
frame.pack()
bottomframe = Frame(root)
bottomframe.pack( side = BOTTOM )
redbutton = Button(frame, text = 'Red', fg='red')
redbutton.pack( side = LEFT)
greenbutton = Button(frame, text = 'Brown', fg='brown')
greenbutton.pack( side = LEFT )
bluebutton = Button(frame, text = 'Blue', fg='blue')
bluebutton.pack( side = LEFT )
```

```
blackbutton = Button(bottomframe, text='Black', fg='black')  
blackbutton.pack( side = BOTTOM)  
root.mainloop()
```





Day-17

Q.1 Multiple Choice Question

1. Essential thing to create a window screen using tkinter python?
a. call tk() function
b. create a button
c. To define a geometry
d. To define the frame.
2. fg in tkinter widget is stands for ?
a. foreground
b. background
c. forgap
d. front group.
3. For user Entry data, which widget we use in tkinter ?
a. Entry.
b. Text.
c. Label.
d. Message.
4. How the place() function put the widget on the screen ?
a. According to x,y coordinate
b. According to row and column wise
c. According to left, right,up,down
d. According to random position.
5. To change the property of the widget after the declaration of widget, what is used?
a. mainloop() function
b. config() function
c. pack() function
d. title() function.

Q. 2 Write a GUI Interface to print the greatest of three numbers.

Q.3 Write a GUI Interface to accept the Person Name and Age. Print the eligibility of person for senior citizen discount , if age is more than or equal to 60.

Q.4 Write a GUI Interface to print the factorial of entered number.

Q.5 Write a GUI interface to accept a list of three names separated by comma in One Textbox. And then split the names and display the name in upper case in 3 different labels.

Q.6 Write a GUI interface to accept the marks of 3 subjects and print the percentage and grade.

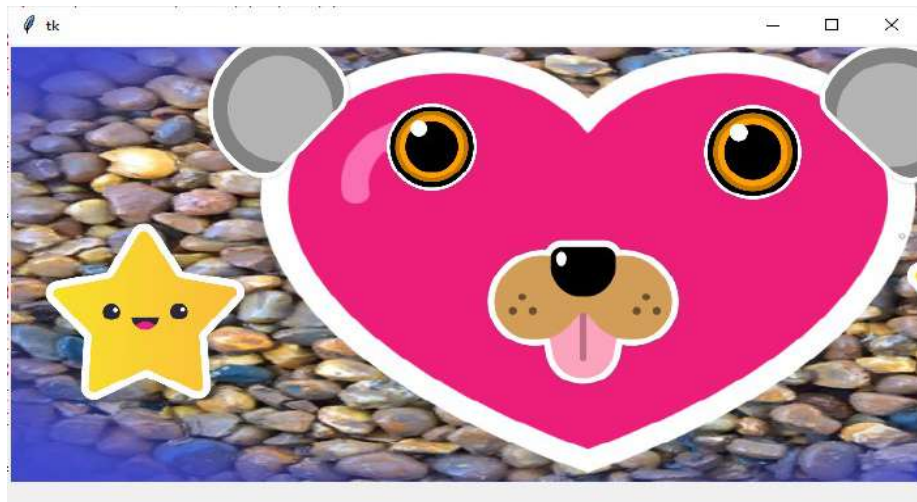
0 - 49	- Grade – F
50 - 74	- Grade – B
75 - 100	- Grade - A

Widgets

Image on Label

```
#Using the tkinter PhotoImage
from tkinter import *
root = Tk()
root.geometry("700x500")

img= PhotoImage(file="d:\\aaa.png")
Label(root, image=img).grid(row=1)
root.mainloop()
```



Example – Image on Label

```
from tkinter import *
#python -m pip install pillow
from PIL import Image , ImageTk
root = Tk()
root.geometry("700x500")
logo=Image.open( "d:\\aaa.png").resize( (100,100), Image.BOX )
```

```
img= ImageTk.PhotoImage( logo)
Label(root, height=300, width=300, image=img).grid(row=1)
root.mainloop()
```



Listbox

It offers a list to the user from which the user can accept any number of options.
The general syntax is:

```
w = Listbox(master, option=value)
```

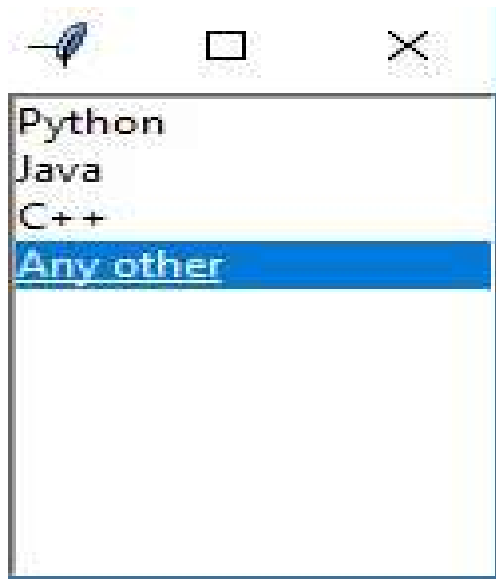
master is the parameter used to represent the parent window.

- **highlightcolor:** To set the color of the focus highlight when widget has to be focused.
- **bg:** to set the normal background color.
- **bd:** to set the border width in pixels.
- **font:** to set the font on the button label.
- **image:** to set the image on the widget.
- **width:** to set the width of the widget.
- **height:** to set the height of the widget.

Example

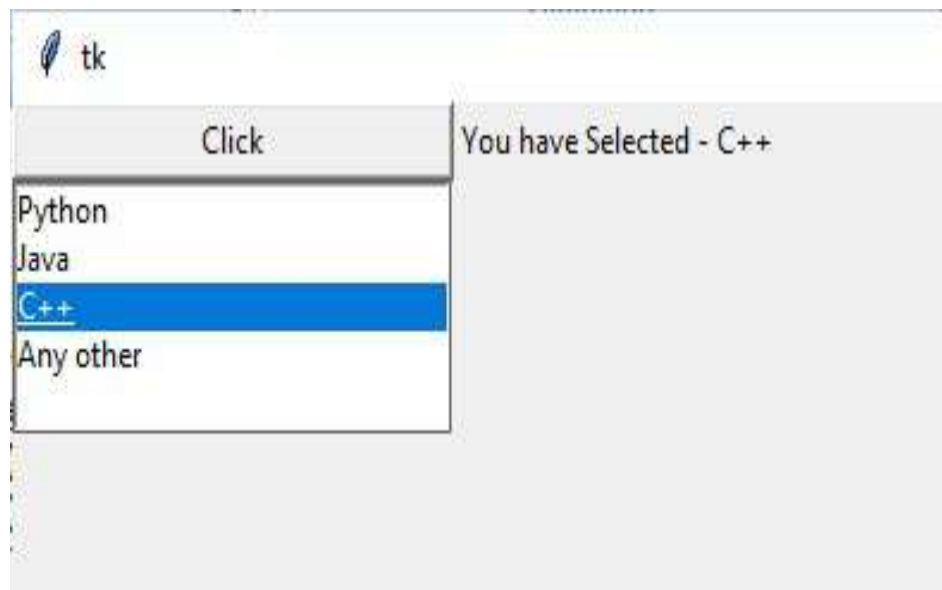
```
from tkinter import *
top = Tk()
Lb = Listbox(top)
Lb.insert(1, 'Python')
Lb.insert(2, 'Java')
Lb.insert(3, 'C++')
```

```
Lb.insert(4, 'Any other')
Lb.pack()
top.mainloop()
```

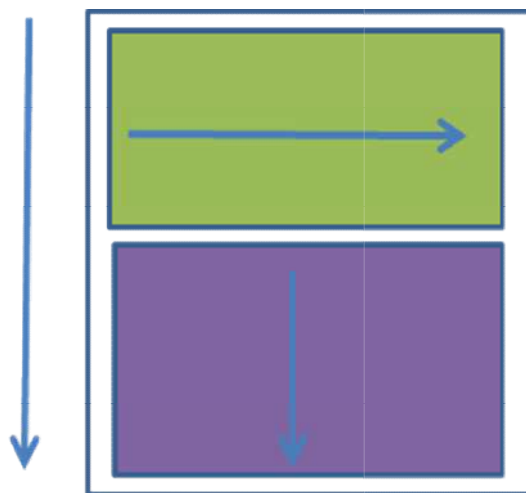


Listbox - Fetching Selected Value

```
# Show the selected value in Label
from tkinter import *
def show( ):
    v=Lb.get(Lb.curselection())
    Lb1.set("You have Selected - " + v)
    return
w = Tk()
#.geometry("window width x height + position right + down")
w.geometry("700x400+300+300")
Lb = Listbox(w,width=30, height=5)
Lb.insert(1, 'Python')
Lb.insert(2, 'Java')
Lb.insert(3, 'C++')
Lb.insert(4, 'Any other')
Lb.grid( row=1 )
Lb1=StringVar()
b1=Button(w, text="Click",width=25, command=show).grid(row=0)
lb1=Label(w, textvariable=Lb1).grid( row=0, column=1)
w.mainloop()
```



PanedWindow



```
from tkinter import *
def show():
    v=var2.get()
    s=var1.get()
    if v==1: var3.set(s.upper())
    if v==2: var3.set(s.lower())
# main tkinter window
```

```
root = Tk()
var1=StringVar()
var2=IntVar()
var3=StringVar()
# panedwindow object
pwM = PanedWindow(orient ='vertical')
pwHTop = PanedWindow(orient ='horizontal')
pwHBot = PanedWindow(orient ='vertical')
pwM.add(pwHTop)
pwM.add(pwHBot)
L1=Label(pwHTop, text='Enter Name ')
L1.pack(side = LEFT)
pwHTop.add(L1)
E1=Entry(pwHTop, textvariable=var1)
E1.pack(side = LEFT)
pwHTop.add(E1)
R1 = Radiobutton(pwHBot, text ="Upper Case!", value=1, variable=var2)
R1.pack(side = TOP)
R2 = Radiobutton(pwHBot, text ="Lower Case!", value=2, variable=var2)
R2.pack(side = TOP)
pwHBot.add(R1)
pwHBot.add(R2)
top=Button(pwHBot, text ="Click Me !\nI'm a Button", command=show)
pwHBot.add(top)
L2=Label(pwHBot, text='Enter Name ', textvariable=var3)
L2.pack(side = LEFT)
pwHBot.add(L2)
pwM.pack(fill = BOTH, expand = True)
mainloop()
```

Menu

It is used to create all kinds of menus used by the application.
The general syntax is:

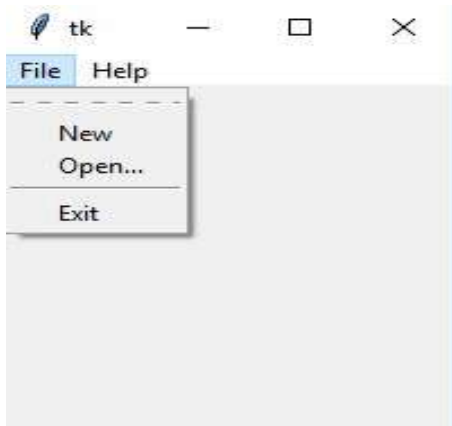
```
w = Menu(master, option=value)
```

master is the parameter used to represent the parent window.

- **title:** To set the title of the widget.
- **activebackground:** to set the background color when widget is under the cursor.
- **activeforeground:** to set the foreground color when widget is under the cursor.
- **bg:** to set the normal background color.
- **command:** to call a function.
- **font:** to set the font on the button label.
- **image:** to set the image on the widget.

Example

```
from tkinter import *
root = Tk()
menu = Menu(root)
root.config(menu=menu)
filemenu = Menu(menu)
menu.add_cascade(label='File', menu=filemenu)
filemenu.add_command(label='New')
filemenu.add_command(label='Open...')
filemenu.add_separator()
filemenu.add_command(label='Exit', command=root.quit)
helpmenu = Menu(menu)
menu.add_cascade(label='Help', menu=helpmenu)
helpmenu.add_command(label='About')
mainloop()
```



Example-2 for Menu

```

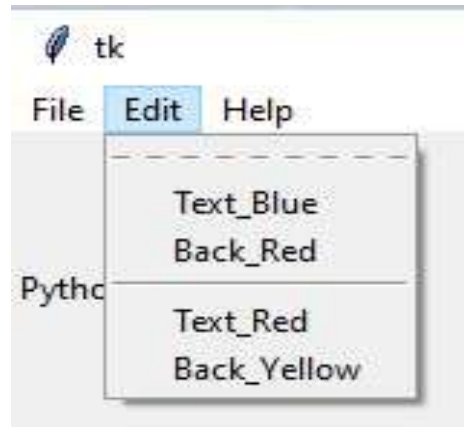
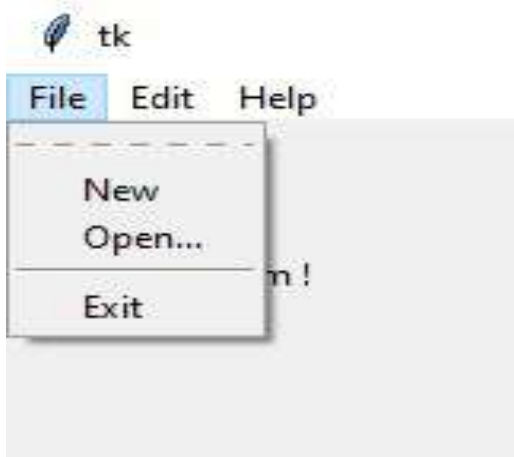
from tkinter import *
def show( menu ):
    if menu=='E_Text_B':  L1.config( fg='blue')
    if menu=='E_Text_R':  L1.config( fg='red')
    if menu=='E_Back_R':  L1.config( bg='red')
    if menu=='E_Back_Y':  L1.config( bg='yellow')
    return

root = Tk()
root.geometry("700x400")
menu = Menu(root)
root.config(menu=menu)
filemenu = Menu(menu)
menu.add_cascade(label='File', menu=filemenu)
filemenu.add_command(label='New')
filemenu.add_command(label='Open...')
filemenu.add_separator()
filemenu.add_command(label='Exit', command=root.quit)
editmenu = Menu(menu)
menu.add_cascade(label='Edit', menu=editmenu)
editmenu.add_command(label='Text_Blue', command=lambda: show('E_Text_B'))
editmenu.add_command(label='Back_Red', command=lambda: show('E_Back_R'))
editmenu.add_separator()
editmenu.add_command(label='Text_Red', command=lambda: show('E_Text_R'))
editmenu.add_command(label='Back_Yellow', command=lambda: show('E_Back_Y'))
helpmenu = Menu(menu)
menu.add_cascade(label='Help', menu=helpmenu)
helpmenu.add_command(label='About')
Frame(root,bd=5, height=50).grid(row=1)
L1=Label(root, text="Python Program !")

```



```
L1.grid(row=4)
root.mainloop()
mainloop()
```



Text

To edit a multi-line text and format the way it has to be displayed.
The general syntax is:

```
w =Text(master, option=value)
```

- **highlightcolor:** To set the color of the focus highlight when widget has to be focused.
- **insertbackground:** To set the background of the widget.
- **bg:** to set the normal background color.
- **font:** to set the font on the button label.
- **image:** to set the image on the widget.
- **width:** to set the width of the widget.

- **height:** to set the height of the widget.

Example

```
from tkinter import *
root = Tk()
T = Text(root, height=2, width=30)
T.pack()
T.insert(END, 'NIELIT Gorakhpur\nPython Programming\n')
mainloop()
```

Events

Mouse Events

<Enter>—The mouse pointer entered the widget.

<Leave>—The mouse pointer left the widget.

<Button-1>,<ButtonPress-1>, or

<1>—A mouse button is pressed over the widget.

<B1-Motion>—The mouse is moved, with mouse button 1 being held down.

<ButtonRelease-1>—Button 1 was released.

<Double-Button-1>—Button 1 was double-clicked.

Keyboard Events

<Key>—The user has pressed any key.

<Control-Up>— The user pressed the Control key, while pressing the Up arrow.

<Return>— The user pressed the Enter key.

<Escape>— The user pressed the Esc key.

The same concept can also be applied for all the other special keys found in the keyboard, including:

F1, F2, F3, F4, F5, F6, F7, F8, F9, F10, F11, F12, Num_Lock, Scroll_Lock, Caps_Lock, Print, Insert, Delete, Pause, Prior (Page Up), Next (Page Down), BackSpace, Tab, Cancel (Break), Control_L (any Control key), Alt_L (any Alt key), Shift_L (any Shift key), End, Home, Up, Down, Left, and Right

Events and Binds

A Tkinter application runs most of its time inside an event loop, which is entered via the `mainloop` method. It waiting for events to happen. Events can be key presses or mouse operations by the user.

Tkinter provides a mechanism to let the programmer deal with events. For each widget, it's possible to bind Python functions and methods to an event.

`widget.bind(event, handler)`

If the defined event occurs in the widget, the "handler" function is called with an event object. describing the event.

Example

```
from tkinter import *

def hello(event):
    print("Single Click, Button-1")

def quit(event):
    print("Double Click, so let's stop")
    import sys; sys.exit()

widget = Button(None, text='Mouse Clicks')
widget.pack()
widget.bind('<Button-1>', hello)
widget.bind('<Double-1>', quit)
widget.mainloop()
```

Day-18

Q.1 Multiple Choice Question

1. From which keyword we import the Tkinter in program?
 - a. call
 - b. from
 - c. import
 - d. All of the above

2. How pack() function works on tkinter widget ?
 - a. According to x,y coordinate
 - b. According to row and column vise
 - c. According to left,right,up,down
 - d. None of the above

3. How the grid() function put the widget on the screen ?
 - a. According to x,y coordinate
 - b. According to row and column vise
 - c. According to left,right,up,down
 - d. None of the above

4. Tkinter tool in python provide the
 - a. Database
 - b. OS commands
 - c. GUI
 - d. All of the above

5. Which of the following is not a geometry managers are available in Tkinter?
 - a. .pack()
 - b. .grid()
 - c. .place()
 - d. .flex()

6. Which one is not a valid Tkinter widgets?
 - a. Label
 - b. Button
 - c. Entry
 - d. WebBrowser



Q.2 Write a GUI program to show the images selected by the user .

```
#Using the tkinter PhotoImage
from tkinter import *
from tkinter import filedialog as fd

def load():
    f= fd.askopenfilename()
    v1.set(f)
    return

def show():
    img= PhotoImage(file=v1.get())
    L=Label(w, image=img)
    L.grid(row=3,column=0)
    L.image=img
    return

w = Tk()
w.geometry("700x500")

v1=StringVar()
Label(w, text='FileName').grid(row=1,column=0)
Entry(w,text=v1).grid(row=1,column=1)
Button(w, text='Load', command=load).grid(row=2,column=2)
Button(w, text='Show', command=show).grid(row=2,column=3)

w.mainloop()
```



Q.3 Write a GUI program to show the graph based on user selection on the data available in CSV file.

```
#Using the tkinter PhotoImage
from tkinter import *
from tkinter import filedialog as fd
import pandas as pd
import matplotlib.pyplot as plt

def load():
    f= fd.askopenfilename()
    f1.set(f)
    return

def plotgraph():
    df=pd.read_csv( f1.get())
    fig=plt.figure()
    if v1.get()==1 : plt.bar( df[ df.columns[0]] , df [ df.columns[1]] )
    if v1.get()==2 : plt.plot(df[ df.columns[0]] , df [ df.columns[1]])
    if v1.get()==3 : plt.scatter(df[ df.columns[0]] , df [ df.columns[1]])
    plt.xticks(rotation=15)
    plt.savefig('d:/g1.png')

def show():
    plotgraph()
    img= PhotoImage(file='d:/g1.png')
    L=Label(w, image=img)
    L.grid(row=5,column=4)
    L.image=img
    return

w = Tk()
w.geometry("700x500")

v1=IntVar()
f1=StringVar()
v1.set(1)

Label(w, text='FileName').grid(row=1,column=0)
Entry(w,text=f1).grid(row=1,column=1)
Button(w, text='Load CSV', command=load).grid(row=2,column=2)

Radiobutton(w, text='Bar', variable=v1, value=1).grid(row=3, column=1)
Radiobutton(w, text='Line', variable=v1, value=2).grid(row=3, column=2)
Radiobutton(w, text='Scatter', variable=v1, value=3).grid(row=3, column=3)

Button(w, text='Show Graph', command=show).grid(row=4,column=3)

w.mainloop()
```

Data Science

- Data Science deals with the extraction of knowledge from large set of data, that may be structured or unstructured.
- Large volume of data source are
 - Scientific Experiments
 - Internet of Things
 - E-commerce
 - Financial Transactions,
 - Bank/credit transaction
 - Social Network

Big Data

- Big Data are large and complex data sets which are difficult for traditional methods to store , access and analyze.
- However, lot of potential values are hidden in this data. This makes it valuable for organizations.
- Big Data technologies and approaches are used to drive values out of data in a efficient ways.
- Types of Data
 - a) Structured – Fields / Tables / columns eg- Spreadsheet, RDBMS
 - b) Semi-Structured – Tags to separate items eg – XML, HTML
 - c) Unstructured – No field/attributes. Eg. email , articles

Facts About Data

- As compared to 2005, 300 times i.e. 40 Zettabytes (1ZB= 10^{21} bytes) of data will be generated by 2020.
- By 2011, healthcare sector has a data of 161 Billion Gigabytes
- 400 Million tweets are sent by about 200 million active users per day
- Each month, more than 4 Billion hours of video streaming is done by the users.
- 30 Billion different types of contents are shared every month by the user.
- It is reported that about 27% of data is inaccurate and so 1 in 3 business idealists or leaders don't trust the information on which they are making decisions.

Data Science Fields

- **Artificial Intelligence** - Getting machines to do, what humans are good in.
- **Machine Learning** – Using algorithm to train and predict on data.
- **Deep Learning** – It is a type of **machine learning** that has networks capable of **learning** unsupervised from data that is unstructured or unlabeled. Also known as **deep neural learning** or **deep neural network**.

Data Science Applications

- **Business** – From car design to pizza delivery, businesses are using data science to optimize their operations and better meet their customer expectations.
- **Health Care**- Health care units are managing the electronic health records, which helps in better point-of-care decisions.
- **Urban Leaving** – Urban informatics deals with challenges, world facing due to growing cities. e.g Traffic Management
- **Web Search Engine**: One of the reasons why search engines like google, bing etc work so well is because the system has learnt how to rank pages through a complex learning algorithm.
- **Photo tagging Applications**- Be it facebook or any other photo tagging application, the ability to tag friends makes it even more happening. It is all possible because of a face recognition algorithm that runs behind the application.
- **Spam Detector**-Our mail agent like Gmail or Hotmail does a lot of hard work for us in classifying the mails and moving the spam mails to spam folder. This is again achieved by a spam classifier running in the back end of mail application.

Data Science - Python Libraries Support

- **NumPy**- It is a library for adding support for large, multi dimensional arrays and matrices, along with a large collection of mathematical functions to operate on these arrays.
- **Scikits** – It is machine learning library that has various classification, regression and clustering algorithms including support vector machines, random forest, gradient boosting and k-means and is designed to interoperate with the python numerical and scientific libraries Numpy and SciPy.
- **Pandas**– It is a software library for data manipulation and analysis. It offers data structures and operations for manipulating numerical tables and time series.
- **Matplotlib**- It is a library for plotting various type of charts for data visualisation.
- **TensorFlow** – It is a library for machine learning applications such as neural networks.
- **Keras**– It is a neural network library. It works on top of TensorFlow and focus on enabling fast experimentation.

Data Science - Steps

1. **Data Gathering** - Put the necessary systems in place to gather data. Data collected may be fragmented and scattered.
2. **Data Preparation** - Clean and format the messy data sets, to make it usable for analysis. This includes managing missing values, error in data collection, data formatting, normalization.
3. **Exploration** - Understand the structure of data, by using the clustering algorithms and visualizing methods - scatter plot, bar graphs.

4. **Model Building** - Explore the variety of models (Random Forest, SVM, Neural Network, K-nearest Neighbors etc) on the data sets and identify and develop the model which fit for the problem.
5. **Model Validation** – Analyze the prediction accuracy of the model based on evaluation matrices.
6. **Model Deployment** – Deploy the model. Tweak and improve it based on feedback.

Data Preparation

1. For achieving better results from the applied model in Machine Learning projects the format of the data has to be in a proper manner. Some specified Machine Learning model needs information in a specified format
2. For example, Random Forest algorithm does not support null values, therefore to execute random forest algorithm null values have to be managed from the original raw data set.
3. Another aspect is that data set should be formatted in such a way that more than one Machine Learning and Deep Learning algorithms are executed in one data set, and best out of them is chosen.

Data Preparation - Steps

- Selecting data objects and attributes for the analysis.
- Creating/changing the attributes.
- Normally following steps are taken before it is ready to be used for machine learning models
 - Importing the libraries
 - Importing the Dataset
 - Handling of Missing Data
 - Handling of Categorical Data
 - Splitting the dataset into training and testing datasets
 - Feature Scaling

Dataset - Filename – property_data.csv

	PID	ST_NUM	ST_NAME	OWN_OCCUPIED	NUM_BEDROOMS	NUM_BATH	SQ_FT
0	100001000.0	104.0	PUTNAM	Y	3	1	1000
1	100002000.0	197.0	LEXINGTON	N	3	1.5	--
2	100003000.0	NaN	LEXINGTON	N	NaN	1	850
3	100004000.0	201.0	BERKELEY	12	1	NaN	700
4	NaN	203.0	BERKELEY	Y	3	2	1600
5	100006000.0	207.0	BERKELEY	Y	NaN	1	800
6	100007000.0	NaN	WASHINGTON	NaN	2	HURLEY	950
7	100008000.0	213.0	TREMONT	Y	--	1	NaN
8	100009000.0	215.0	TREMONT	Y	na	2	1800

Find out the following questions:

- What are the features?
- What are the expected types (int, float, string, boolean)?
- Standard Missing data (values that Pandas can detect)?
- Non Standard Missing Data (can't easily detect with Pandas)?

Loading Dataset

```
# Importing libraries
import pandas as pd
import numpy as np
# Read csv file into a pandas dataframe
df = pd.read_csv("d:\data\property_data.csv")
# Show the first 5 rows of the dataframe
print(df.head())
```

	PID	ST_NUM	ST_NAME	OWN_OCCUPIED	NUM_BEDROOMS	NUM_BATH	SQ_FT
0	100001000.0	104.0	PUTNAM	Y	3	1	1000
1	100002000.0	197.0	LEXINGTON	N	3	1.5	--
2	100003000.0	NaN	LEXINGTON	N	NaN	1	850
3	100004000.0	201.0	BERKELEY	12	1	NaN	700
4	NaN	203.0	BERKELEY	Y	3	2	1600

Dataset Features

```
# Structure of the Dataframe
print(df.info())
# Column of the Dataframe
print(df.columns)
# Row and Column of the Dataframe
print(df.columns)
```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9 entries, 0 to 8
Data columns (total 7 columns):
#   Column                Non-Null Count  Dtype
---  -
0   PID                   8 non-null     float64
1   ST_NUM                7 non-null     float64
2   ST_NAME               9 non-null     object
3   OWN_OCCUPIED          8 non-null     object
4   NUM_BEDROOMS          7 non-null     object
5   NUM_BATH              8 non-null     object
6   SQ_FT                8 non-null     object
dtypes: float64(2), object(5)
memory usage: 388.0+ bytes
None

```

4 5

what are my features?

ST_NUM : Street number
ST_NAME : Street name
OWN_OCCUPIED : Is the residence owner occupied
NUM_BEDROOMS : Number of bedrooms

Expected Data Types

what are the expected types?

ST_NUM : float or int... some sort of numeric type
ST_NAME : string
OWN_OCCUPIED : string... Y ("Yes") or N ("No")
NUM_BEDROOMS : float or int, a numeric type

	PID	ST_NUM	ST_NAME	OWN_OCCUPIED	NUM_BEDROOMS	NUM_BATH	SQ_FT
0	100001000.0	104.0	PUTNAM	Y	3	1	1000
1	100002000.0	197.0	LEXINGTON	N	3	1.5	--
2	100003000.0	NaN	LEXINGTON	N	NaN	1	850
3	100004000.0	201.0	BERKELEY	12	1	NaN	700
4	NaN	203.0	BERKELEY	Y	3	2	1600

Standard Missing Data

Standard Missing data

(values that Pandas can detect)?

Finding Missing data in ST_NUM Column

```

print(df['ST_NUM'])
print(df['ST_NUM'].isnull())

```

Standard Missing Data

none

NaN

```
0      104.0
1      197.0
2         NaN
3      201.0
4      203.0
5      207.0
6         NaN
7      213.0
8      215.0
Name: ST_NUM,
```

```
0      False
1      False
2       True
3      False
4      False
5      False
6       True
7      False
8      False
Name: ST_NUM,
```

Non Standard Missing Data

Non Standard Missing data (values that Pandas can detect)?

Finding Missing data in ST_NUM Column

```
print(df['NUM_BEDROOMS'])
```

```
print(df['NUM_BEDROOMS'].isnull( ))
```

Non standard Missing Data

n/a

NA

—

na

```

0      3
1      3
2    NaN
3      1
4      3
5    NaN
6      2
7     --
8     na
Name: NUM_BEDROOMS,

```

```

0    False
1    False
2     True
3    False
4    False
5     True
6    False
7    False
8    False
Name: NUM_BEDROOMS,

```

Non Standard Missing Data

Non Standard Missing data

(Pandas can detect now)

Importing libraries

```
import pandas as pd
```

```
import numpy as np
```

Making a list of missing value types

```
missing_values = [ "n/a", "na", "--" ]
```

```
df = pd.read_csv("d:\data\property_data.csv", na_values = missing_values)
```

Finding Missing data in NUM_BEDROOMS Column

```
print(df['NUM_BEDROOMS'])
```

```
print(df['NUM_BEDROOMS'].isnull( ))
```

```

0      3.0
1      3.0
2      NaN
3      1.0
4      3.0
5      NaN
6      2.0
7      NaN
8      NaN
Name: NUM_BEDROOMS,

```

```

0      False
1      False
2       True
3      False
4      False
5       True
6      False
7       True
8       True
Name: NUM_BEDROOMS,

```

Unexpected Missing Values

1. If Column expected to be a string, but having a numeric type data, is technically a missing value.

Owner Occupied should clearly be a string (Y or N), so this numeric type should be a missing value

```

# Finding Missing data in OWN_OCCUPIED Column
print(df['OWN_OCCUPIED'])
print("\n")
print(df['OWN_OCCUPIED'].isnull())

```

```

0      Y
1      N
2      N
3     12
4      Y
5      Y
6     NaN
7      Y
8      Y
Name: OWN_OCCUPIED,

```

```

0     False
1     False
2     False
3     False
4     False
5     False
6     True
7     False
8     False
Name: OWN_OCCUPIED,

```

Unexpected Missing Values

Convert the unexpected value to NaN

```

cnt=0
for row in df['OWN_OCCUPIED']:
    try:
        int(row)
        df.loc[cnt, 'OWN_OCCUPIED']=np.nan
    except ValueError:
        pass
    cnt+=1

```

```

0      Y
1      N
2      N
3     NaN
4      Y
5      Y
6     NaN
7      Y
8      Y
Name: OWN_OCCUPIED,

```

```

0     False
1     False
2     False
3      True
4     False
5     False
6      True
7     False
8     False
Name: OWN_OCCUPIED,

```

Summarizing Missing Values

Total missing values for each Column

```
print(df.isnull().sum())
```

Any missing values?

```
print(df.isnull().values.any())
```

#Total number of missing values

```
print(df.isnull().sum().sum())
```



```

PID                1
ST_NUM             2
ST_NAME            0
OWN_OCCUPIED       2
NUM_BEDROOMS       4
NUM_BATH            1
SQ_FT              2
dtype: int64

```

```

-----
-
True
-----
-

12

```

Replace Missing Value

Imputation – To replace the missing value with mean/median/mode of all the value for that column.

Replacing Missing Values

```

# Replace missing values with a number
df['ST_NUM'].fillna(125, inplace=True)
#Common to replace missing values is using a median
median = df['NUM_BEDROOMS'].median( )
df['NUM_BEDROOMS'].fillna(median, inplace=True)
# Total missing values for each feature
print( df.isnull().sum( ))

```

```

PID                1
ST_NUM             2
ST_NAME            0
OWN_OCCUPIED       2
NUM_BEDROOMS       4
NUM_BATH            1
SQ_FT              2
dtype: int64

```

```

PID                1
ST_NUM             0
ST_NAME            0
OWN_OCCUPIED       2
NUM_BEDROOMS       0
NUM_BATH            1
SQ_FT              2
dtype: int64

```

```

# Replace using median
median = df['SQ_FT'].median()
df['SQ_FT'].fillna(median, inplace=True)
# Replace using median
df['NUM_BATH'].loc[df['NUM_BATH']=='HURLEY']=1
median = df['NUM_BATH'].median()
df['NUM_BATH'].fillna(median, inplace=True)
df['OWN_OCCUPIED'].fillna('Y', inplace=True)
df['PID'].fillna(100005000, inplace=True)
# Total missing values for each feature
print( df.isnull().sum( ))

```

```

PID                0
ST_NUM             0
ST_NAME            0
OWN_OCCUPIED       0
NUM_BEDROOMS       0
NUM_BATH            0
SQ_FT              0
dtype: int64

```

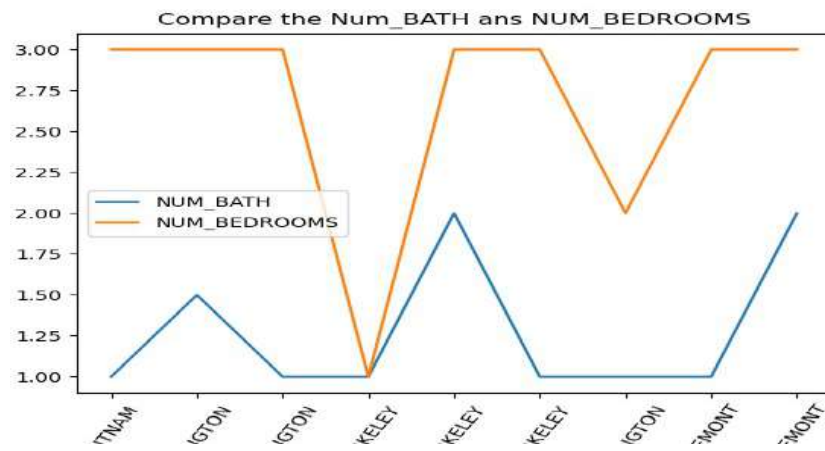
Visualisation in Pandas

```

df['NUM_BATH']=df['NUM_BATH'].astype('float',copy=False)
print(df.info( ))
ax=plt.gca( )
plt.title("Compare the Num_BATH ans NUM_BEDROOMS")
df.plot( x='ST_NAME' , y='NUM_BATH', kind='line',ax=ax)
df.plot( x='ST_NAME' , y='NUM_BEDROOMS', kind='line',ax=ax)

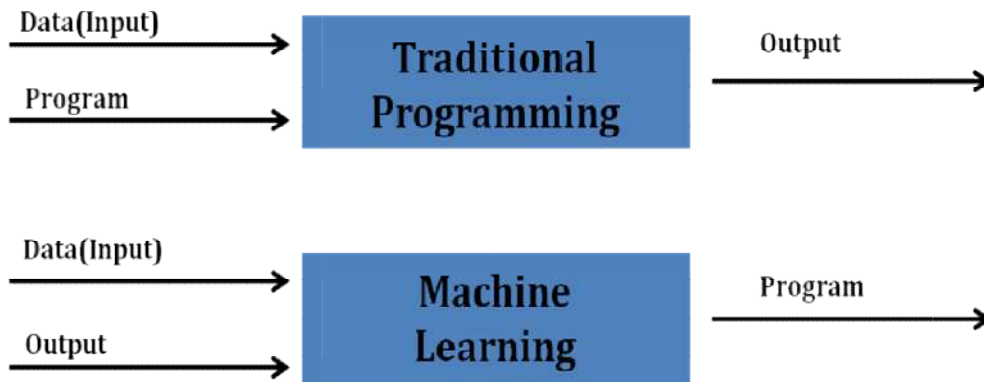
```

```
plt.xticks(rotation=90)  
plt.show()
```



Machine Learning

- **Traditional Programming** : We feed in DATA (Input) + PROGRAM (logic), run it on machine and get output.
- **Machine Learning** : We feed in DATA(Input) + Output, run it on machine during training and the machine creates its own program(logic), which can be evaluated while testing.



Machine Learning - Classification

- Machine learning implementations are classified into three major categories, depending on the nature of the learning “signal” or “response” available to a learning system which are as follows:-
 - a) Supervised learning
 - b) Unsupervised learning
 - c) Reinforcement learning

Machine Learning - Classification

1. **Supervised learning** –
 - When an algorithm learns from example data and associated target responses that can consist of numeric values or string labels, such as classes or tags, in order to later predict the correct response when posed with new examples.
 - The teacher provides good examples for the student to memorize, and the student then derives general rules from these specific examples.
2. **Unsupervised learning**
 - Whereas when an algorithm learns from plain examples without any associated response, leaving to the algorithm to determine the data patterns on its own.
 - It resembles the methods humans use to figure out that certain objects or events are from the same class, by observing the degree of similarity between objects.
3. **Reinforcement learning** –
 - When you present the algorithm with examples that lack labels, as in unsupervised learning. However, you can provide an example with positive or

negative feedback according to the solution . In the human world, it is just like learning by trial and error.

- Errors help you learn because they have a penalty added (cost, loss of time, regret, pain, and so on), teaching you that a certain course of action is less likely to succeed than others.

Supervised vs Unsupervised learning

Example 1:

- You get a bunch of photos **with information about what is on them**.
- Then you train a model to recognize new photos.

Example 2:

- You have a bunch of molecules and information about which are drugs.
- Then you train a model to answer whether a new molecule is also a drug.

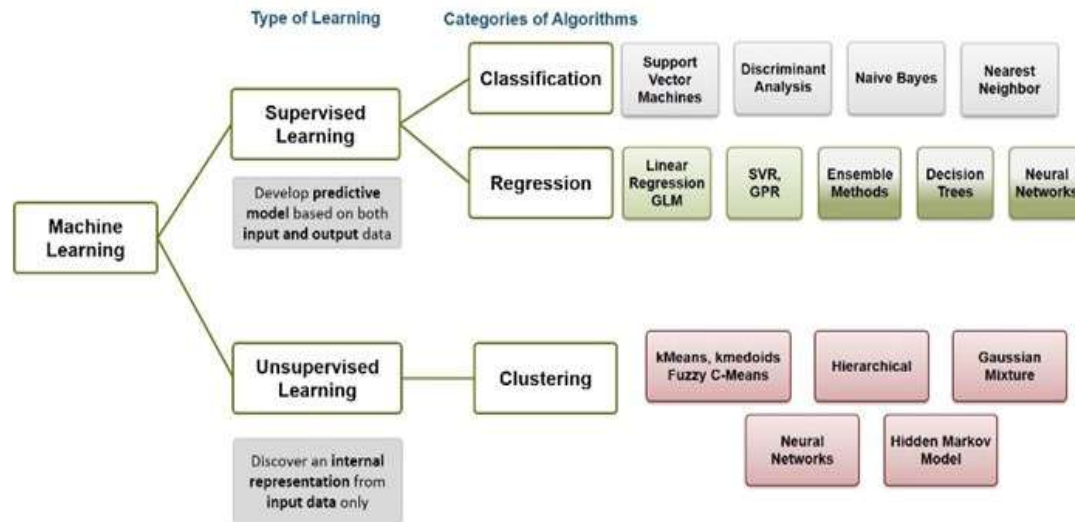
Example 1:

- You have a bunch of photos of 6 people
- **No information about who is on which one.**
- You want to **divide** this dataset into 6 piles, each with the photos of one individual.

Example 2:

- You have molecules, part of them are drugs and part are not.
- But you do not know which are which.
- You want the algorithm to discover the drugs.

Category of Algorithms

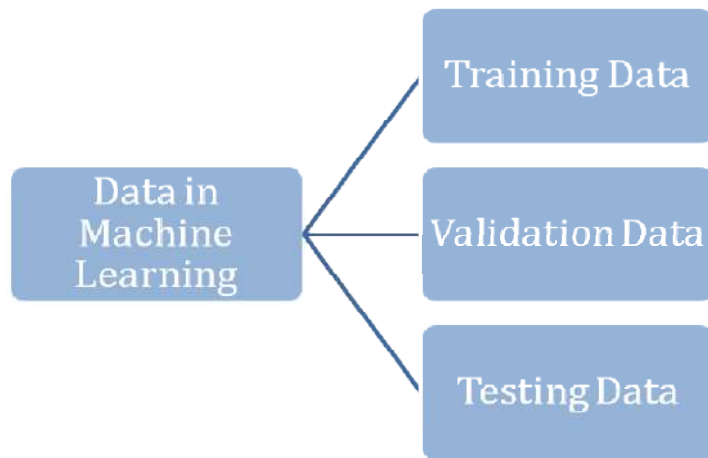


Machine Learning - Working

- Divide the input data into
 - training,
 - cross-validation and
 - test sets.
- The ratio between the respective sets must be 6:2:2
- Building models with suitable algorithms and techniques on the training set.
- Testing our conceptualized model with data which was not fed to the model at the time of training and evaluating its performance using metrics such as F1 score, precision and recall.

Machine Learning - Splitting Data

1. **Training Data:** The part of data we use to train our model. This is the data which your model actually sees(both input and output) and learn from.
2. **Validation Data:** The part of data is used to do a frequent evaluation of model, fit on training dataset along with improving involved parameters (initially set parameters before the model begins learning). This data plays it's part when the model is actually training.
3. **Testing Data:** Once model is completely trained, testing data provides the unbiased evaluation. When we feed in the inputs of Testing data, our model will predict some values(without seeing actual output). After prediction, we evaluate our model by comparing it with actual output present in the testing data. This is how we evaluate and see how much our model has learned from the experiences feed in as training data, set at the time of training.



Machine Learning - Examples

1. **Siri**(part of Apple Inc.'s iOS, watchOS, macOS, and tvOS operating systems),
2. **Google Now** (a feature of Google Search offering predictive cards with information and daily updates in the Google app for Android and iOS.),
3. **Cortana** (Cortana is a virtual assistant created by Microsoft for Windows 10) are intelligent digital personal assistants on the platforms like iOS, Android and Windows respectively.
4. They help to find relevant information when requested using voice. For instance, for answering queries like 'What's the temperature today?' or 'What is the way to the nearest supermarket' etc. and the assistant will react by searching information, transferring that information from the phone or sending commands to various other applications.

Machine Learning - Implementation

Loading the CSV File into DataFrame

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
df=pd.read_csv("d:\\datascience\\programs\\htwgtMale.csv")
print(df.head( ))
```

```
print(df.info( ))
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 38 entries, 0 to 37
```

```
Data columns (total 2 columns):
```

```
#   Column  Non-Null Count  Dtype
```

```
---  ---
```

```
0   Height    38 non-null   int64
```

```
1   Weight    38 non-null   int64
```

```
dtypes: int64(2)
```

```
memory usage: 672.0 bytes
```

```
None
```

	Height	Weight
0	105	12
1	110	10
2	119	21
3	120	28
4	132	33

Extracting the required Columns from the DataFrame

```
#Returns a Numpy representation of the DataFrame.
```

```
X=df["Height"].values
```

```
#Only the values in the DataFrame will be returned, the axes labels will be removed.",
```

```
Y=df["Weight"].values
```

```
print('type(X):', type(X))
```

```
print('X.shape:', X.shape, 'Y.shape:', Y.shape)
```

```
#Reshaping the array from (1 X N) to (N X 1)
```

```
X=X.reshape(-1,1)
```

```
Y=Y.reshape(-1,1)
```

```
print('Changed X.shape:', X.shape)
```

Output

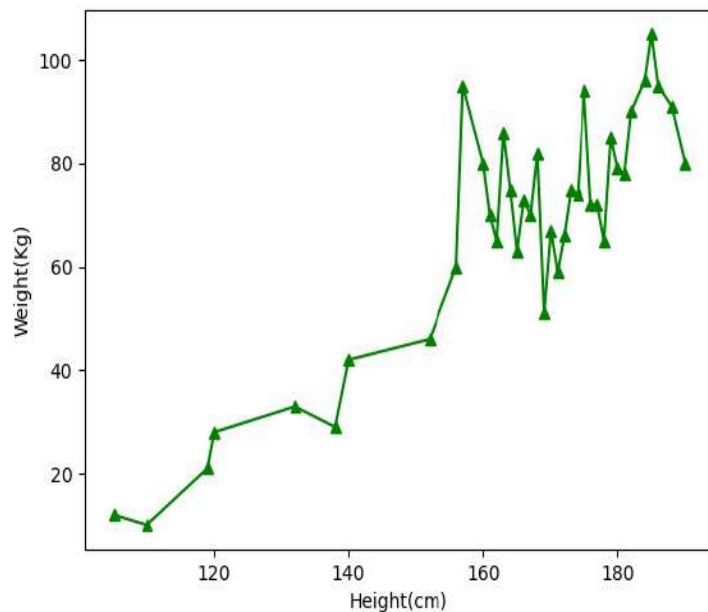
```
type(X): <class 'numpy.ndarray'>
```

```
X.shape: (38,) y.shape: (38,)
```

```
Changed X.shape: (38, 1)
```


Plotting the Trend in the Data

```
#Plot the Trend of Original Data
plt.plot(X, Y, c='g', marker="^", label='Original Data')
plt.xlabel('Height(cm)')
plt.ylabel('Weight(Kg)')
plt.show( )
```



Splitting the Data for Training & Testing

```
#Split the Data into Training & Validation Data
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, random_state=0, test_size=.3)
print('type(X_train):', type(X_train))
print('Shapes of X_train, X_test, Y_train, Y_test are:')
print(X_train.shape, X_test.shape, Y_train.shape, Y_test.shape )
```

Output

Data Original Shape

X.shape: (38,) y.shape: (38,)

Shapes of X_train, X_test, Y_train, Y_test are:

(26, 1) (12, 1) (26,) (12,)

Training the Model - LinearRegression

```
lr=LinearRegression()

#Training of Data
lr.fit(X_train, Y_train )

#Prediction
Y_pred=lr.predict(X_test)
print('Shape of y_pred is:')
print(Y_pred.shape)

print("Test Val<--Vs-->Predicted Val")
for i in range(len(Y_test)):
    print(Y_test[i], "<--Vs-->", Y_pred[i])
```

Output

```
Test Val<--Vs-->Predicted Val
[66] <--Vs--> [74.83920048]
[85] <--Vs--> [81.63826886]
[73] <--Vs--> [69.01142759]
[63] <--Vs--> [68.0401321]
[70] <--Vs--> [64.15495017]
[67] <--Vs--> [72.89660952]
[72] <--Vs--> [78.72438241]
[80] <--Vs--> [63.18365469]
[94] <--Vs--> [77.75308693]
[21] <--Vs--> [23.36053991]
[78] <--Vs--> [83.58085982]
[96] <--Vs--> [86.49474627]
```

Plotting the Model - LinearRegression

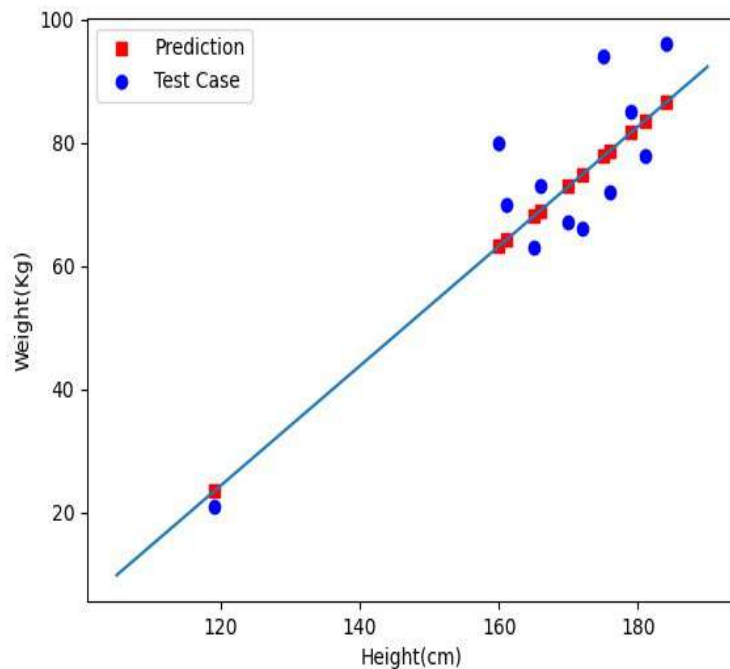
```
#We can plot X vs y using the equation of the form  $y=m*x+c$ 
#lr.coef_ gives m and lr.intercept_ gives
m=lr.coef_
c=lr.intercept_
print('Slope =',m,'y-intercept=',c)

yCalc= m*X+c
plt.plot(X,yCalc)
```

```
plt.scatter(X_test, Y_pred, c='r', marker="s", label='Prediction')
plt.scatter(X_test, Y_test, c='b', marker="o", label='Test Case')
```

```
plt.legend(loc='upper left')
plt.xlabel('Height(cm)')
plt.ylabel('Weight(Kg)')
plt.show( )
```

Output



Accuracy of the Model - LinearRegression

#R-squared is a statistical measure of how close the data are to the fitted regression line.

```
print("R-Square Value on Training Data: ", lr.score(X_train, Y_train))
print("R-Square Valueon Testing Data : ", lr.score(X_test, Y_test))
```

Output

R-Square Value on Training Data: 0.7858953710613918

R-Square Valueon Testing Data : 0.7746542312358535

Data in CSV File

Height	Weight
105	12
110	10
119	21
120	28
132	33
138	29
140	42
152	46
156	60
157	95
160	80
161	70
162	65
163	86
164	75
165	63
166	73
167	70
168	82
169	51
170	67
171	59
172	66
173	75
174	74
175	94
176	72
177	72
178	65
179	85
180	79
181	78
182	90
184	96
185	105
186	95
188	91
190	80

Complete Program

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

df=pd.read_csv("d:\\datascience\\programs\\htwgtMale.csv")

'''
print(df.head())
print(df.info())
'''

X=df["Height"].values

#Above Returns a Numpy representation of the DataFrame.
#Only the values in the DataFrame will be returned, the axes labels will be removed.\n",

Y=df["Weight"].values
print('type(X):', type(X))
print('X.shape:', X.shape, 'y.shape:',Y.shape)

#Reshaping the array from (1 X N) to (N X 1)
X=X.reshape(-1,1)
print('Changed X.shape:',X.shape)
Y=Y.reshape(-1,1)

'''
#Plot the Trend of Original Data
plt.plot(X, Y, c='g', marker="^", label='Original Data')
plt.xlabel('Height(cm)')
plt.ylabel('Weight(Kg)')
plt.show()
'''

#Split the Data into Training & Validation Data
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, random_state=0,test_size=.3)
print('type(X_train):',type(X_train))

print('Shapes of X_train, X_test, Y_train, Y_test are:')
```

```

print(X_train.shape,X_test.shape, Y_train.shape, Y_test.shape )

lr=LinearRegression()

#Training of Data
lr.fit(X_train, Y_train )

#Prediction
Y_pred=lr.predict(X_test)

print('Shape of y_pred is:')
print(Y_pred.shape)

print("Test Val<--Vs-->Predicted Val")
for i in range(len(Y_test)):
    print(Y_test[i],"<--Vs-->",Y_pred[i])

'''
#We can plot X vs y using the equation of the form  $y=m*x+c$ 
#lr.coef_ gives m and lr.intercept_ gives
m=lr.coef_
c=lr.intercept_
print('Slope =',m,'y-intercept=',c)

yCalc= m*X+c

plt.plot(X,yCalc)

plt.scatter(X_test, Y_pred, c='r', marker='s', label='Prediction')
plt.scatter(X_test, Y_test, c='b', marker='o', label='Test Case')
plt.legend(loc='upper left')
plt.xlabel('Height(cm)')
plt.ylabel('Weight(Kg)')
plt.show()

'''
#R-squared is a statistical measure of how close the data are to the fitted regression
line.\n",
print("R-Square Value on Training Data: ", lr.score(X_train, Y_train))
print("R-Square Valueon Testing Data : ", lr.score(X_test, Y_test))

```

```
import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

df=pd.read_csv('d:/datacsv/d16-covid.csv')

#1

plt.plot('State','May-20', data=df) #matplotlib

sns.lineplot(x='State', y='May-20', data=df) # Seaborn

plt.show()

#2

plt.bar('State','May-20', data=df)

sns.barplot(x='State', y='May-20', data=df)

plt.show()

#3

plt.bar('State','May-20', data=df)

plt.plot('State','May-20', data=df)

plt.show()

sns.lineplot(x='State', y='May-20', data=df)

sns.barplot(x='State', y='May-20', data=df)

plt.show()

#4

df1=pd.melt(df, id_vars='State', value_vars=['May-20', 'Jun-20'])

sns.lineplot(x='State', y='value', data=df1)

plt.show()
```

```
sns.lineplot(x='State', y='value', hue='variable', data=df1)
```

```
plt.show()
```

```
sns.barplot(x='State', y='value', hue='variable', data=df1)
```

```
plt.show()
```

```
sns.barplot(x='State', y='value', data=df1)
```

```
plt.show()
```

```
#5
```

```
df1=pd.melt(df, id_vars='State', value_vars=['May-20', 'Jun-20'])
```

```
g=sns.FacetGrid(data=df1, col='variable')
```

```
g.map( sns.scatterplot, 'State', 'value')
```

```
g.set_xticklabels(rotation=30)
```

```
#plt.xticks(rotation=30)
```

```
plt.subplot(121)
```

```
plt.plot('State', 'May-20', data=df)
```

```
plt.title('May-20')
```

```
plt.subplot(122)
```

```
plt.plot('State', 'Jun-20', data=df)
```

```
plt.title('Jun-20')
```

```
plt.show()
```

```
#6
```

```
df1=pd.melt(df, id_vars='State', value_vars=['May-20', 'Jun-20'])
```

```
g=sns.JointGrid(x='State', y='value', hue='variable', data=df1)
```

```
g.plot_joint(sns.lineplot)
```

```
g.plot_joint(sns.barplot)
```

```
plt.show()
```



```
import pandas as pd
```

```
import matplotlib.pyplot as plt
```

```
import seaborn as sns
```

```
df=pd.read_csv('d:/datacsv/Loan-applicant-details.csv')
```

```
#1
```

```
sns.scatterplot(x='ApplicantIncome', y='LoanAmount', data=df)
```

```
plt.show()
```

```
#2
```

```
sns.scatterplot(x='ApplicantIncome', y='LoanAmount', hue='Gender',data=df)
```

```
plt.show()
```