1st Note

Python — Full Zero to Advance

by wscube.Tech — YT

Use NotePad For First
Then Vs Code editor OR Any Other

** Introduction to python.

⇒ Our agenda:-

* what is python?
* History of python?
* Features of python
* Application of python
* Companies using python

* what is python?

⇒ Python is an interpreted, object-oriented, high-level programming language with dynamic semantics.

⇒ Python syntax are easy compared to other to other languages.

* History of python.

→ Python was introduced by Guido van Rossum in 1989.

→ First version of python released in 1991.

→ In 1994, Python 1.0 was released with new features which map, filter, lambda.

→ Python 2.x also add some more feature like compressions, garbage collection system.

→ After Python 2.x, Python 3.x released in 2008

* Features of python
    (1) Easy to learn and Implement
    (2) It is open source.
    (3) Broad standard Library
    (4) Cross platform
    (5) Work on Interpreter logic
    (6) Multi-paradigm language
    (7) High Level programming
    (8) Extendable language.
    (9) Expressive programming language
    (10) GUI programming support.

* Application of python.
    - Network programming
    - Data Analysis
    - Robotics
    - Website and application development.
    - Desktop Application
    - Games Development
    - web scraping
    - Data Visualization
    - Scientific Calculation
    - Machine Learning and Artifical Intelligence
    - 3D Application Development
    - Audio and videos software Development.

* Companies using python
  - Google
  - Netflix – video buffering
  - Facebook / meta – Handle Large data
  - Youtube – recommand system
  - Nasa – scientific calculator
  - Dropbox
  - Torrent – Pair to pair
  - Instagram – crood video processing
  - USA – Nation Security Agency
  - Quora

▲ Installation of Python in windows

* How to Download and install python on windows

→ Download from python.org

Practice

* How to Install Pycharm for python on windows.

⇒ Pycharm is code editor.

⇒ Pycharm download = search in browser.

⇒ Create Folder

⇒ Create file name :- demo.py

```
print("welcome to Wscubetech")
```

▲ Create the First Python "Hello World" Program using Print Function.

* Create First Hello world Program in Python using Print( ) Function

→ Create file with name first.py

```
Print ("Hello World")
Print (20+50)
```

→ Create file with ---

```
Print ("Hello world, welcome to wscubeTech")
Print (10>5)
Print (5 & 10)
```

▲ Create and Use Variables in python

* Variable in Python

→ Create file name as varcheck.py

```
a=10
~~Print (a)~~ b=10
Print (a,b)
~~Print(id(a), id(b))~~
Print ('a')
Print ('a',b)
Print(id(a),id(b))

c = "Hello"
Print(c)
```

1a = 20 ✗
a1 = 20 ✓
a b = 35 ✗
a_b = 35 ✓
@a = 20 ✗
_a = 20 ✓

String Concatenation in Python

* String Concatenation in Python

→ Sum of string. Use + for concatenation.

→ Create file name :- string.c.py

```
a = "Hello"
b = " Wscubetech"
# Print (a+b)
Print (a+" " +b)
c = 20
Print (c+30)
# Print (a+c) — # string+ variable is error.
```

~~Arithmetic~~ Operations in Python

* Operator in Python (Arithmetic operators)

Operator	Name	Example
+	Addition	x+y
−	Substraction	x−y
*	Multiplication	x*y
/	Division	x/y
%	Modulus	x%y
**	Exponents	x**y
//	Floor division	x//y

→ Create file name :- opt.py

```
a = 10
b = 20
```

```
print(a+b)    # Addition
print(a-b)    # Substraction
print(a*b)    # multiplication
print(a/b)    # Division.
print(b%3)    # Find reminder using
print(a%3)      modulus.
print(5**2)   # 5*5   # Exponents.
print(5**3)   # 5*5*5
print(10/3)   # floor division gives
print(10//3)    value in integer. It
                displaces after decimal
                values.
```

7. Assigment Operators

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x+3
-=	x -= 3	x = x-3

→ Create new file :- aop.py

```
x = 5
print(x)
x = x+5
print(x)
x += 5
print(x)
```

```
x = x - 5
print(x)
x - = 5
print(x)
```

* Comparison Operators in Python

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

→ Print function works on boolen algebra.

→ Create file name: cop.py

```
# x = 10
# y = 20
# print(x == y) → False
# x = 10
# y = 10
# print(x == y) → True
# print(x != y) → False
# x = 10
# y = 20
# print(x != y) → True
# print(x > y) → False
# print(x < y) → True
```

```
# print(x >= y) → False
# print(y >= x) → True
```

* Logical Operators in Python

Operators	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, return false, if the result is true	not(x<5 and x<10)

→ Create file name: 1.py

```
# x=10
# y=20
#print(x==10. and x<y)
#Print(x==10 and x<y and x==y)
# Print(x==10 or x<y or x==y)
# Print(x!=10 or x>y or x!=y)
# Print(not x!=10)
# Print(not(x!=10 and x>y and x==y))
```

* Membership Operators in Python

Operator	Description	Example
in	Returns True if a sequence with the specified value is present in 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

→ Create file name :- m.py

```
~~h = "Hello"~~
~~print ('h'~~
# string1 = "Hello"
# print ('h' in string1)
# print ('H' in string1)
# print ('o' in string1)
# list = [10, 20, 30, 40]
# print (50 in list)
~~# print (30 in list)~~
# print (50 not in list)
# print (30 in list)
# print (30 not in list)
```

* Identity Operators in Python

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

⇒ Create file name :- idn.py

```
# x = 10
# y = 10
~~# print (x is y) ⇒ (x == y)~~
# print (x is y, x == y)
# print (x != y, x is not y)
```

```
# x = 10
# x = 20
# print(x is y, x == y)
# print(x is not y, x != y)
```

* Bitwise Operators in python
→ 0, 1 → format.

Operator	Name	Description
&	AND	x & y
|	OR	x | y
^	NOR	x ^ y

Binary Truth Table

1 = TRUE 0 = FALSE

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

→ Create a file name :- bop.py

```
# x = 10
# y = 8
```

```
# Print(bin(16))
# Print(bin(y))
# Print(x & y)   # Print(bin(x & y))
# Print(x & y, bin(x & y))
# |
# Print(bin(x|y))
# Print(bin(x   # Print(x|y, bin(x|y))
# Print(bin(x^y))
# Print (x^y, bin(x^y))
```

Practice

Data Types in Python - Part ⇒ 1

⇒ Data types list :-

* Numeric
  - Integers
  - Float
  - Complex Numbers
* Sequence Type
  - String
  - List
  - Tuple
* Dictionary
* Set

* Mutable and Immutable Data Types
⇒ Mutable object can change its state or contents and ~~imp~~ immutable object cannot.

Mutable Data Types:
- List
- Dictionary
- byte array

Immutable Data Types:
- ~~int~~ int
- Float
- Complex
- String
- Tuple
- Set

* Numeric
- Integers
- Float
- Complex numbers

```
a = 5 print(a, "is of type", type(a))
    # 5 is of type <class 'int'>
a = 2.0 print(a, "is of type", type(a))
    # 2.0 is of type <class 'float'>
a = 1+2j print(a, "is of type", type(a))
    # is of type <class 'complex'>
```

→ Create file name :- dtlist.py

```
a = 5
Print(a)
Print(a, type(a)) # <class 'int'>
a = 5.5
Print(a, type(a)) # <class 'float'>
a = 2+5j
Print(a, type(a)) # <class 'complex'>
```

* Sequence type

- String
- List
- Tuple

- String :-

A string is a collection of one or more characters put in a single quote, double-quote or triple quote.
Multi-line strings can be denoted using triple quotes, ''' or """

```
S = "This is a string"
print(S)
S = '''A multiline string'''
print(S)
```

→ Create file name :- string.py

```
S = "Hello@123"
Print(type(S))
```

```
s = '''
        Hello
        welcome to uscubetech
    '''
print(s)
print(type(s))

s = '10'
print(type(s))
```

- **List:-**

List is an ordered sequence of items.

→ It is one of the most used datatype in python and is very flexible.

→ [ ]

e.g. a = [1, 2.2, 'ws']

→ Create file name :- list1.py

```
l = [10, 'ws', 5.5]
print(l, type(l))
l[2] = 10 # update value function
print(l, type(l))
```

- **Tuple:-**

Tuple is an ordered sequence of items same as a list.

→ It is defined within parentheses () where items are separated by commas.

e.g. t = (5, 'Program', 1+3j)

→ Create a file name :- tuple.py

```
t = (10, 20, 'Hello')
print(t, type(t)) # Tuple.
~~t = (10)~~ t = (10)
print(t, type(t)) # integer.
```

* Dictionary :-

Dictionary is an unordered collection of key-value pairs.

In python, ~~dice~~ dictionaries are defined within braces { } with each item being a pair in the form key : value

e.g.

```
d = {1: 'value', 'key': 2}
print(type(d))
<class 'dict'>
```

→ Create a file name : dict.py.

```
d = {
    ~~'name'~~ : 'course_name': 'Python',
    'course_duration': '2 month'
    }
print(d, type(d))
print(d['course_name'])
```

* Set :-

A set is an unordered collection of items.

→ Every set element is unique (no duplicates) and must be immutable (cannot be changed)

→ { }

e.g.

```
my_set = {1,2,3}
print(my_set)
```

→ Create file name: set.py

```
s = {10, 20, 30, 10}
print(s)
print(s, type(s)) # remove it automatically if value repeated
```

Practice

▲ User Input and Type Casting in Python.

→ Key Points:-

* input( )
* int( )
* float( )
* eval( )

} Type Casting function.

* input( )

→ Create file name:- input.py

```
a = input("Enter the value :- ")
print(a)

b = input("Enter the value 1: ") # string value
c = input("Enter the value 2: ")
print(a+b)
```

* int( ) – Convert value in the integer
→ Create file name: in.input.py

```
a= int(input("Enter The value1"))
b= int(input("Enter the value2"))
print(a+b)
```

* float( ) - convert value in float
→ Create file name: flo.input.py

```
a= float(input("Enter the value1"))
b= float(input("Enter the value2"))
print(a+b)
```

* eval( ) – It take all type of value.
→ Create the file name: eval.input.py

```
a= eval(input("Enter the value1")
b=eval(input("Enter the value2")
print(a+b)
```

▲ If, If Else and If Elif Else Conditional Statements in Python.
⇒ Python Conditional Statements:

Key Points:–
(1) If statement
(2) If else statement
(3) If elif else statement

(1) If statement → display only true value

→ if [conditional expression]:
[statement(s) to execute]

Example:-

```
a = 10
if a % 2 == 0:
    print ("Even Number");
```

→ Create file name: (statement.py

```
a = int (input ("Enter the value :- "))
if a % 2 == 0:
    print (a, "Even Number")

b = int (input ("Enter your Age : "))
if a > 18:
    print ("your are teenager : ")
```

(2) if else statement → display True and false.

→ if [conditional expression]:
[statement to execute]
else:
[alternate statement to execute]

Example:-

```
a = 10
if a % 2 == 0
    print ("Even Number")
```

```
else:
    print("odd Number")
```

→ Create filename :- c1.py

```
~~a = int(input("Enter the value : "))~~
~~a = int (input( a, "Even number"))~~
~~if a%2==0:~~
    ~~print(a,~~
a = int (input("Enter the value : "))
if a%2==0:
    print(a, "Even number")
    ~~print(b, "Odd number")~~
else:
    print(a, "Odd number")

~~b = int(input("Enter~~
age = int(input("Enter your age : "))
if age >= 18:
    print("you are eligible for vote.")
else:
    print("you are not eligible for vote")
    print("Welcome to new Journey")
```

(3) if elif else statement

⇒ if [condition #1]:
    [statement #1]
elif [condition #2]:
    [statement #2]

```
elif [condition #3]:
    [Statement #3]
else:
    [statement when if and elif(s) are
        false]
```

Example:-

```
percentage = 55
if percentage >= 60:
    print("First division")
elif percentage >= 48:
    print("Second division")
elif percentage >= 35:
    print("Third division")
else:
    print("Fail")
```

⇒ Create a file name : c2.py

```
Percentage = int(input("Enter the value"))
if percentage >= 60:
    print("First division")
elif percentage >= 48:
    print("Second division")
elif percentage >= 35:
    print("Third division")
else:
    print("Fail")
```

if, elif, else statement

▲ How to Build a simple Calculator in Python

⇒ Create file name: Calculator.py.

```
print('''
    + Add
    - subtract
    * Multiply
    / Divide
''')
```

Best method For conditional

```
num1 = int(input("Enter the value: 1 "))
num2 = int(input("Enter the value: 2 "))
opr = input(" Enter the opr... ")
if opr == "+":
    print(num1+num2)
elif opr == "-":
    print(num1-num2)
elif opr == "*":
    print(num1*num2)
elif opr == "/":
    print(num1/num2)
else:
    print("invalid opr...")
```

⇒ Create file name:- calculator1.py

```
print('''  + Add
         - subtract
         * multiply
         / divide''')
```

Never use This method OR if only

```
no1 = int(input("Enter the value1: "))
no2 = int(input("Enter the value 2: "))
opr = input("Enter the opr---(+,-,*,/)")

if opr == "+":
    print(no1 + no2)
if opr == "-":
    print(no1 - no2)
if opr == "*":
    print(no1 * no2)
if opr == "/":
    print(no1/no2)
if opr != "+" and opr != "-" and opr != "*"
    and opr != "/":
    print("Invalid opr---")
```

Practice

▲ For Loop with Range() in python.

→ Create a file name: floop.py

~~range(5)~~ ~~#start~~	#range(5)
#Range()-range(5)	#range(1,6)
→ start = 0	→ start = 1
→ condition < 5	→ condition < 6
→ increment 1.	→ increment 1
→ 0,1,2,3,4	→ 1,2,3,4,5

```
#range(1,6,2)
```
→ start = 1
→ condition <6
→ increament 2
→ 1,3,5

```
#for n in range(5):
    print(n)
#output
    0
    1
    2
    3
    4
```

```
#For n in range(1,6):
    print(n)
#for n in range(1,6,2):
    print(n)
#For n in range(5):
    print("welcome")
#for a in range(1,11)
    print(a)
    print(2*a)
```

→ Create filename: f1.py

```
#for a in range(1,11):
    print("2*",a,2*a)
```

```
*Reverse case range
    #range(10,0,-1)
    #for n in range
    (10,0,-1):
        print(n)
#range(1
#for n in range(10,0,-2):
    print(n)
#for n in range(10,4,-2):
    print(n)
```

▲ While Loop in python
→ Create a file name: w.py

→ 'while' loop contains:
- start
- Condition — i=i+1
- increment/decrement

```
i=1
while i<=10:
    print("welcome to wscubetech")
    i=i+1

print(i) #output=11
```

Reverse run

```
a=10
while a>=1:
    print(a, "hello")
    a=a-1
print(a). #Output=0
```

String = Function

▲ String Indexing and String slicing in python.

* String in python
→ A string is a sequence of characters.

→ strings can be created by enclosing characters inside a single quote or double-quotes

→ ~~String~~ Triple quotes can be used represent multiline strings.

* String indexing

-17	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
W	E	L	C	O	M	E		T	O		W	S	C	U	B	E
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

→ Create file : S1.py

```
w= "Welcome to wscube"
print(w[6])
print(w[7])
print(w[-10])
print(w[16])
print(w[-1])
```

* String slicing

→ Create file : S2.py

```
w="welcome to wscube"
print(w[0:7]) → Gives data upto 6.
print(w[0::2]) → increament by 2 value.
print(w[2])
print(w[0:7:2])
print(w[-1:-10:-2]) → Reverse decreasement.
print(w[-1::-1]) → Reverse decreasement
```

▲ String iteration in python
→ for loop is used for iteration

→ Create file name: s3.py

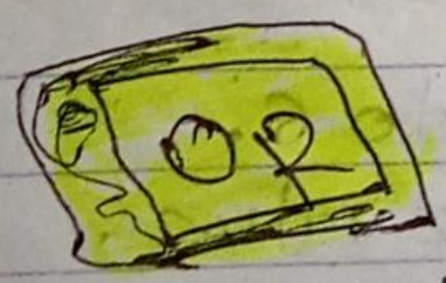
```
w="welcome to wscubetech"
t=len(w) ↔length of string with space count
print(t) #output=21
for a in range(t): #21
    print(w[a])
#w[0]=w — output
#w[0]=e — output
```

- - - - - same as ①

Reverse iteration

→ create filename: s4.py

```
w="welcome to wscubetech"
t=len(w) → length of string with space count.
print(t) #output=21
for a in range(t-1,-1,-1): #21
    print(w[a])
```



```
w="welcome to wscubetech"
for a in w:
    print(a)
```

▲ Lower, Upper, Title and Capitalize String functions in python

→ Python string Function :-

* lower( ) — lower all letter
* upper( ) — upper all letter
* title ( )
* Capitalize( )

→ Create: sfun.py

```
w="Welcome to Wscubetech"
n=w.lower()
print(n)

y="WElcome to WscuBetecH"
a=y.upper()
print(a)

z="WELCOME TO WSCUBETECH"
t=z.title()
print(t)

w="Welcome to Wscubetech"
c=w.capitalize()
```

▲ Find, Index, Isalpha, isdigit and isalnum String functions in python

⇒ Python String Functions :-

* find()
* index()
* isalpha()
* isdigit()
* isalnum()

→ Gives result/output in True and false

⇒ Create file name : sfun1.py

```
w="Welcome"
print(w.find('e'))
~~print(w.find('e',1))~~
print(w.find('e',2)) - starting from 2.
print(w.find('z',2)) → result in -1.
                       → index not present
```

```
w="Welcome"
print(w.index('e')
print(w.index('e',2))
print(w.index('z',3)) #output error
print(w.index('z')) # output error
```

⇒ Create file name : ssfun2.py

```
W="Welcome"
print(w.isalpha()) #output True

w="Welcome123"
print(w.isalpha()) #output false.
```

~~print(w)~~

~~w = "welcome123"~~

```
print(w.isdigit()) #output false
w = "123"
print(w.isdigit()) #output true
print(w.isalpha()) #output false

w = "welcome123"
print(w.isalnum()) #output true

w = "welcome 123"
print (w.isalnum()) #output false
```

▲ Ord() and chr() functions in python

* Python chr() ~~and ord()~~ Function
⇒ This takes in an integer i and converts it to a character c, so it returns a character string.

- Convert integer 65 to ASCII character('A')

```
y = chr(65)
print(type(y),y)
```

output:
<class'str'> A

→ chr() values start form 65 and upto 94 for capital and for small just oposite

Capital letter value start form = 65 to 94
Small letter value start from = 97 to 123

→ Create file name: strfunction.py

```
a=65;
print(chr(a)) # output = A
a= 66;
print (chr(a)) # output = B
a= 67;
print (chr(a)) # output = C
```

* The ord( ) Function
⇒ This takes a single unicode character (string of length 1) and returns an integer, so the format is:

- Convert ASCII unicode character 'A' to 65

```
y= ord('A')
print(type(y),y)

<class 'int'>65;
```

→ Do on : strfunction.py file:-

```
h= 'A'
print(ord(A)) #output = 65
print(ord(a)) #output = 97
```

[small letter start from = 97 - 123]

[capital letter start from = 65 - 91]

⇒ This both takes only one letter or value.

```
P = "Hello"
print(ord(Be)) # output = error
```

→ String takes and convert to numeric value and vice versa.

▲ String Format() Method in Python

* Python string format() method
⇒ mostly used function.
⇒ The format() method formats the specified value(s) and insert them inside the string's placeholder.
⇒ The placeholder is defined using curly brackets: {}. Read more about the placeholders in the placeholder section below.

Example:-

- named indexes:

```
txt1 = "Welcome to {fname} {lname}".format
        (fname = "Wscube", lname = "Tech")
print(txt1) # output = Welcome to Wscube Tech
```

- Numbered indexes:

```
txt2 = "Welcome to {0} {1}".format
        ("Wscube", "Tech")
print(txt2) # output = Welcome to Wscube Tech
```

- Empty placehotdersi

```
txt3 = "Welcome to {} {}".format
        ("Wscube", "Tech")
print(txt3) = Welcome to Wscube Tech
```

→ Create file name :- st.format.py

~~w = "Welcome to Wscubeteeh.format("~~
~~.format("hello", 20);~~

w = "Welcome {} to {} Wscubeteeh".format
("hello", 20);

print(w)
#output = Welcome hello to 20 Wscubeteeh

~~w = "Welcome {0} {1}~~
w = "Welcome {0} to {1} Wscubeteeh".format
("hello", 20);

print(w)
#output = Welcome hello to 20 Wscubeteeh

w = "Welcome {1} to {0} Wscubeteeh".format
("hello", 20);

print(w)
#output = Welcome 20 to hello Wscubeteeh

w = "Welcome {a} to {b} Wscubeteeh".format
(a=30, b=40)
print(w)
#output = Welcome 30 to 40 Wscubeteeh

w = "Welcome {b:11} to {a:5} Wscubeteeh".
format (a=30, b=40)

print (w)
#output = Welcome ---------- 40 to ---30 Wscu
-beteeh

```
w = "Welcome {b:^10} to {a:^8} Wscubetech"
        .format(a=30, b=40)
print(w)
#output = Welcome ----40---- to ---30---
          Wscubtech

w = "Welcome {b:>10} to {a:>8} Wscubetech"
        .format(a=30, b=40)
print(w)
#output = Welcome 40-------- to 30------
          Wscubetech

w = "Welcome {b:<10} to {a:<8} Wscubetech"
        .format(a=30, b=40)
print(w)
#output = Welcome --------40 to -------30
          Wscubetech
```

▲ Unlimated msg using python

```
*Impress-my-gf.py
import pyautogui as pg
import time
print ("Program will run in 5 sec")
time.sleep(5)
for i in range(100):
    pg.write ("i love you")
    time.sleep(0.5)
    pg.press("Enter")
```

2nd Note

Python – Zero to Advance

by WSCubeTech – YT

Lists

▲ List in python → Create a file name :- List1.py
→ List is mutable/changable. Decleared in square bracket and comma seperated

For e.g. -5 -4 -3 -2 -1
l = [1, 2, 3, 4, 5]
0 1 2 3 4

Print(type(l)) # output = class('List')
Print(l[2]) # output = 3
l = [1, 2, 3, [4, 5, 6]]
0 1 2 3

Print(l[3][1]) # output = 5
-4 -3 -2 -1
l = [2, 3, "Hello", [3, 4, 5]]
0 1 2 3

point(l[2]) # output = Hello ⇒ mixed List
point(l[0:2]) # output = [2, 3]
Point(l[1:]) # Output = [3, "Hello", [3, 4, 5]]

l = [1, 2, 3, 4, 5, 6]
point(l[0::2]) # output = [1, 3, 5]

Reverse List — Point(l[-1::-1])
Output = [6, 5, 4, 3, 2, 1]

→ Create a file name: list2.py
a = [10, 30, 40, 50, "Hello"]
print(type(a)) # Output = class - List
print(a[3]) # output = 50

print(a[3],a[4]) #output = 50 Hello
print(a[0:2]) #output = [10, 30]
print(a[0::2]) #output = [10, 40, "Hello"]
print(a[3:]) #output = [50, 'Hello']
print(a[-1::-2]) #output = ['Hello', 40, 10]

Reverse-List

print(a[-1::-1]) #output = ['Hello', 50, 40, 30, 20]

▲ List iteration in python

⇒ Create a file name :- List.it.py

l = [10, 20, 30, ~~40, 60~~ 50, 60]

~~print(l~~ t = len(l) #output = 5

for n in range(t):

print(l[n]

#output = 10
20
30
40
50
60

l[0] = 10
l[1] = 20
l[2] = 30
l[3] = 40
l[4] = 50
l[5] = 60

2nd Method

for a in l:
print(a) output = 10
20
30
40
50
60

Reverse iteration:

```
l = [10,20,30,50,60]
t = len(l) #output = 5
for n in range(t-1,-1,-1)
    print (l[n]) #output = 60
                           50
                           30
                           20
                           10
```

⇒ Create a file name:- list-iteration.py

```
l = [10,20,30,40,60,80,90]
t = len (l) #output = 7
print (t) #output = 7
for a in range (t):
    print (l[a]) #output = 10
                           20
                           30
                           40
                           60
                           80
                           90
print( )
for a in l:
    print(a) #output = 10
                       20
                       30
                       40
                       60
                       80
                       90
```

Reverse:

```
l=[10,20,30,40,60,80,90]
t=len(l) #output=7
for a in range(t-1,-1,-1):
    ~~print(l[a]~~
    print(a)
    print(l[a]) #output= 90
                         80
                         60
                         40
                         30
                         20
                         10
```

Thank You

List Function

[Pop, Remove & Clear] List Method

⇒ *Function for Delete Element from list:-

(1) del
(2) pop()
(3) remove()
(4) Clear()

→ Create a file name: listfun.py

```
l=[20,30,50,60]
print(l)
# del l[1]
# print(l)  #output=[20,50,60]
```

```
l.pop(2)
print(l)  #output=[20,30,60]
print(l.pop(2))  #output=50
print(l)  #output=[20,30,60]
l.remove(60)
print(l)  #output=[20,30,50]
l.clear()
print(l)  #output=[]
```

* Update Element from list:-

```
l=[20,30,40,50]
l[0]=10
print(l)  #output=[10,30,40,50]
```

→ Create a file name :- list.up.py

```
l=[20,30,50,60]
l[0]=90
print(l)  #output=[90,30,50,60]
```

→ Function for update element from list:-

- insert()
- append()
- extends()

→ Go on same file :- list.up.py

```
l.insert(0,10)
print(l)  #output=[10,20,30,50,60]
l.append(70)
print(l)  # output=[10,20,30,50,60,70]
```

→ Filename – list.ext.py

```
l=[20,30,50,60]
n=[50,60]
l.append(n)
print(l) #output =[20,30,50,60,[50,60]]
l.extend(n)
print(l) #output =[20,30,50,60,50,60]
```

▲ List Comprehension in python – Elegant way to create Lists

~~* List Comprehension~~

⇒ list comprehension is an elegant way to define and create lists based on existing lists

⇒ list comprehension is generally more compact and faster than normal functions and loops for creating list.

Syntex of list Comprehension

[expression for item in list]

⇒ Create a file name:- list.com.py

```
l=[ ]

for a in range[1, 101]:
    print(a) #output = 1
                       2,.....
                       100
```

```
q=[ ]
for a in range(1,101):
    q.append(a)
print(q)    #output = [1,2,3,4,....,100]
```

~~n=[m for~~ *List-compression method:-

```
n=[m for m in range(1,101)]
print(n)  #output=[1,2,3,4,.....,100]
```

~~n=[m for m in range(1,101) m%2==0]~~

```
n=[m for m in range(1,101) if m%2==0]
print(n)
```

~~print(n) for~~ #output = odd number
= [2,4,6,....]

```
s="hello"
d=[g for g in s]
prind(d)  #output = store string in list
                  =['h','e','l','l','o']
```

▲ [Pop, Remove and clear] List method

* List-Function
→ Function for Delete element from list:-
* del
* pop()
* remove()
* clear()

→ Create a file name :- list.function.py

```
l=[20,30,50,60]
print(l)          #output = [20,30,50,60]
# del l[1]  # return not delete element.
print(l) # output = [20,50,60]
#l.pop(2)
print(l) #output=[20,30,60]
print(l.pop(2))

# l.remove(60)
print(l) #output=[20,30,50]
l.clear()
print(l) #output=[ ]
```

⇒ Update Element from list:-

```
l=[20,30,40,50]
l[0]=10
print(l) #output=[10,30,40,50]
```

Create a file name :- list.up.py

```
l=[20,30,50,60]
l[0]=90
print(l) #output=[90,30,50,60]
```

* insert()
* append()
* extends()

```
l = [20,30,50,60]
l.insert(0,10)
Print(l) # Output = [10,20,30,50,60]
l.append(70)
print(l) #output = [10,20,30,50,60,70]

m = [20,30,50,60]
n = [50,60]
m.append(n) #
print(m) #output = [20,30,50,60,[50,60]]
m.extend(n)
print(m) #output = [20,30,50,60,50,60]
```

▲ Count, Max, Min, Sort, Reverse and index List function

⇒
* Count()
* Max()
* min()
* Sort()
* reverse()
* index()

⇒ Create a file name :- list_count.py

```
l = [10,20,30,50,10]
a = l.count(10)
print(a) #output = 2

m = max(l)
Print(m) #output = 50
```

```
n=['Hello','world']
o=max(n)
print(o) # output = world
p=min(l)
print(p)  #output=10
r=min(n)
print(r) #output = Hello
# l.sort()
#print(l) #output=[10,10,20,30,50]
# l.reverse()
# #print(l) #output=[10,50,30,20,10]
l.index(20)
print(l) #output=1
```

⇒ Create a file name list.func.py

```
l=[10,50,90,100,200,10,10,10]
c=l.count(10)
print(c) #output=4
d=max(l)
print(d) #output=200
m=['I','miss','you']
e=max(m)
print(e) #output=you
f=min(l)
print(f) #output=10
g=min(m)
print(g) #output=I
```

- l.sort() — ascending order/descending
- print(l) #output=[10,10,10,10,50,90,100,200]

l.reverse() [200, 100, 90, 50, 10, 10, 10, 10]
print (l) #output = ~~[10, 10, 10, 200, 100, 90, 50, 10]~~
l.index (50)
print (l) #output = 3

Zip function - Iterate over 2+ Lists at the Same Time
⇒ Create a file name :- zip-fun.py

```
l = [10, 20, 40, 50]
l1 = [3, 4, 77, 88]
for a, b in zip(l, l1):
    print (a,b) #output = 10  3
                          20  4
                          40  77
                          50  88

l3 = [5, 6, 8, 9, 11, 12]
for c, d in zip(l, l3):
    print (c,d) #output = 10  5
                          20  6
                          40  8
                          50  9

t = len (l)
for h in range(t):
    print (h) #output = 0
                        1
                        2
                        3
                        4
    print(l[h]) #output = 10
                          20
                          40
                          50
```

```
for i in range(t):
    print(l[i], l1[i]) #output = 10  3
                                 20  4
                                 40  77
                                 50  88
```

Python Program to Convert String to a list:

⇒ Create a filename :- prg.str.py

```
# n = input ("Enter the value:")
# print(n) #value => 'welcome to WSCubeTech'

# l = n.split();
# print (l) # output = ['welcome', 'to', 'WSCubeTech']

~~t = [ ]~~
~~for a in range (3):~~
    ~~n = input ("Enter the value:")~~
l = [ ]
for a in range (1, 4):
    n = input ("Enter the value" + str(a) + ":- ")
# print (n) #output = [ Hello
                        Hlo
                        Hii ]

l.append(n)
print (l)  #output = ['red', 'blue', 'green']
```

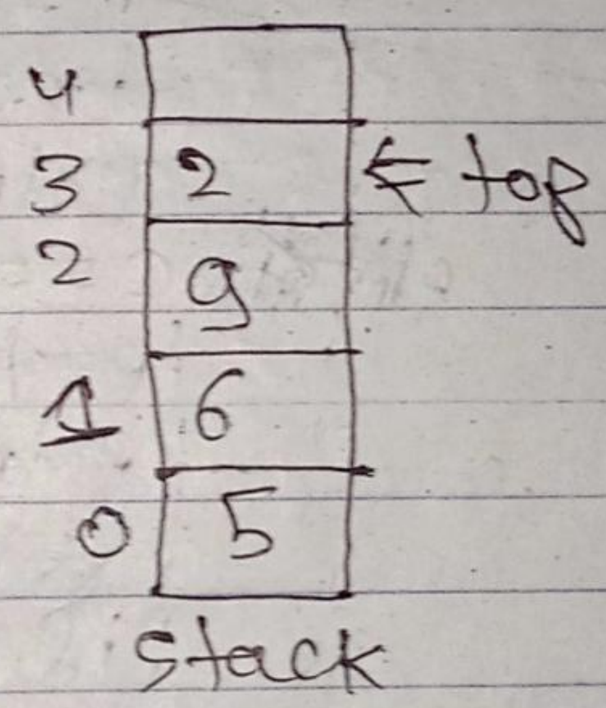
Implement a stack and Queue using a list Data Type

⇒

* stack in python
  ⇒ The stack is a linear data structure
  ⇒ Stores items in a last-in/First-out (LIFO) or First-in/Las-out (FILO) manner.

Stack Operations:-

(1) Push ⇒ inserting an elements
(2) Pop ⇒ Deletion an elements (Last element)
(3) Peek ⇒ Display the last element
(4) Display ⇒ Display list



⇒ ~~Create a file name:- stackops.py~~

~~l = [ ]~~

~~while True:~~

~~c = input("""~~ ~~1 Push Elements~~
~~2 Pop Elements~~
~~3. Peek Element~~
~~4 Display stack~~
~~5~~

⇒ Create a file name :- stackopt.py

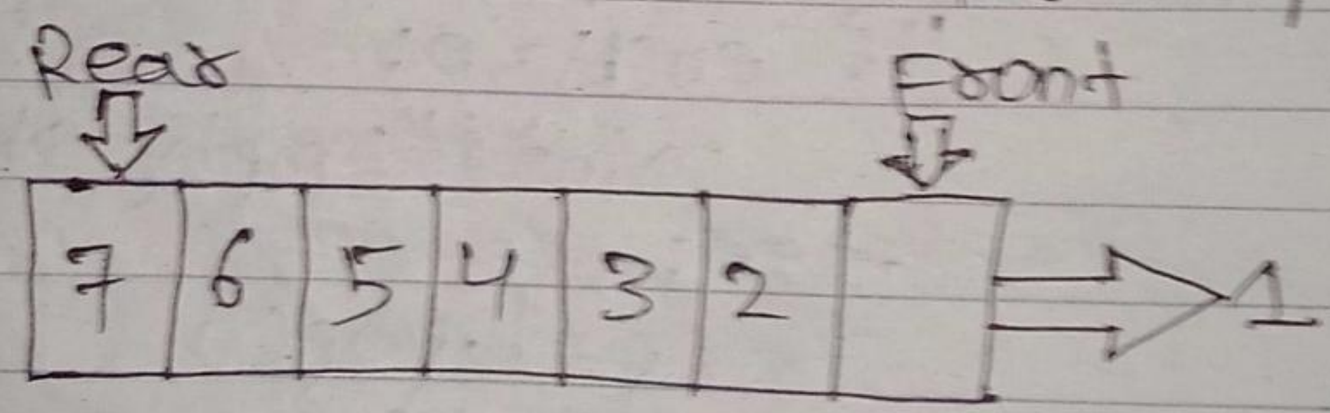
```
l=[ ]
while True:
        c =int(input("""
                        1 Push Elements
                        2 pop Elements
                        3 peek Elements
                        4 Display Elements
                        5 Exit
                        """))
if c==1:
        n=input("Enter the value:");
        l.append(n)
        print(l)
elif c==2:
        if len(l)=0:
                print("Empty stack")
        else:
                p=l.pop()
                print(p)
                print(l)

elif c==3:
        if len(l)=0
                print("Empty stack")
        else:
                print("Last stack value", l[-1])
elif c==4:
        print("Display stack", l)
elif c==5:
        break;
```

```
        else:
            print("Invalid operation")
```

* Queue in python
  → The Queue is a linear data structure.
  → Stores items in First in First out (FIFO) manner.

Queue Operations:
(1) Enqueue: Add an item to the queue.
(2) Dequeue: Removes an item from the queue.
(3) Front: Get the front item from queue
(4) Rear: Get the last item from queue



⇒ Create a file name:- Queue.py

```
l=[]
while True:
    c=int(input('''
                1 Push Elements
                2 pop First Elements
                3 Front Elements
                4 Last Elements
                5 Display Elements
                6 Exit
```

```
if c==1:
    n=input("Enter the value:- ");
    l.append(n)
    print(l)
elif c==2:
    if len(l)==0:
        print("Empty Queue")
    else:
        del l[0]
        print(l)

elif c==3:
    if len(l)==0:
        print("Empty Queue")
    else:
        print("First Queue value:-",l[0])
elif c==4:
    if len(l)==0:
        print("Empty Queue")
    else:
        print("Last Queue value:-",l[-1])

elif c==5:
    print("Display Queue",l)
elif c==6:
    break;
else:
    print("Invalid operator")
```

Array in Python

→ Python does not have built-in support for arrays, but python Lists can be used instead.

* Arrays:—

This page shows you how to use ~~how to use~~ LISTS as ARRAYS, however, to work with arrays in python you will have to import a library, like the Num.py library.

→ Arrays are used to store multiple values in one single variable:

Example:

Create file name:- Array.py

```
cars = ["Ford", "volvo", "BMW"]
print(cars) #output = ["Ford", "volvo", "BMW"]
  x = cars[0]
  print(x) #output = Ford
  y = cars[3]
  print(y) #output = Syntex error

  cars[0] = "Toyota"
  print(cars) #["Toyota", "volvo", "BMW"]

  x = len(cars)
  print(x) #output = 3
for x in cars:
    print(x) #output = Ford
                       volvo
                       BMW
```

cars.append("Honda")
print(cars) #output = ["Ford", "Volvo", "Bmw", "Honda"]

cars.pop(1)
→ delete the second element of the cars array:
print(cars) #output = ["Ford", "BMW"]

> Note:- The list's remove() method only removes the first occurrence of the specified value

*Array methods - Functions:-

Method → Description

(1) append() → Adds an element at the end of the list.

(2) clear() → Removes all the elements from the list.

(3) copy() → Returns a copy of the list.

(4) count() → Returns the numbers of elements with the specified value.

(5) extend() → Add the elements of a list (or any iterable), to the end of the current list

(6) index() → Add an element at the specified position.

(7) pop() → Removes the element at the specified position.

(8) remove() ⟶ Removes the first item with the specified value.

(9) reverse() ⟶ Reverse the order of the list

(10) Sort() ⟶ Sorts the list.

Introduction to Dictionary in python.

⇒ It is unordered data type.

⇒ It work on base key/value

⇒ defined in { }

⇒ It is mutuable.

⇒ Create a file name :- dict.py

```
d = { 'name' : 'Python',
      'fees' : 8000,
      'duration' : '2 months' }
```

Keys ⟶ 'duration', 'fees'

Values ⟶ '2 months', 8000

```
print(d['fees']) #output = 8000
print(d['name']) #output = Python
for n in d:
    print(n) #output = name
                       fees
                       duration
    print(d[n])
                 #output = Python
                           8000
                           2 months
```

⇒ A dictionary is a collection which is unordered. In python dictionaries are written with curly brackets { }.

→ A dictionary has a key: value pair.

→ Create a file name: dict1.py

```
d = {'name': 'Ramesh'
     'surname': 'Sah'
     'stream': 'CS'
     'address': 'saptari'}
```

• Access elements from a dictionary

```
n = d['name']
print(n) #output = Ramesh
```

~~• Iterating through a Dictionary~~

~~print(d[~~

```
print(type(d)) #output = <class-dict>
```

• Iterating Through a Dictionary.

```
for n in d:
    print(n)
    print(d[n]) #output = Ramesh
                          Sah
                          CS
                          Saptari
```

* Dictionary function

• get() → 'get' is used to take out data
• keys() →
• values()
• items()

Create a file name:- dict2.py

```
d = { 'name' : 'rahul',
      'fees' : '10000',
      'duration' : '1 year' }

n = d.get('name')
print(n) # ~~python~~ rahul = output
print(d['name']) #output = python.

for a in d.keys():
    print(a) #output = name
                       fees
                       duration

for b in d.values():
    print(b) #output = rahul
                       10000
                       1 year

for a,b in d.items():
    print(a,b) #output = name rahul
                         fees 10000
                         duration 1 year
```

Create a file name:- dict3.py

```
d = { 'name' : 'python',
      'fees' : '8000',
      'duration' : '2months' }
```

⇒ Delete function

- del
- pop()

} → Keys are used to delete item.

```
# del d['fees']
# print (d) #output = {'name':'python',
                        'duration':'2 months'}

a = d.pop('fees')
print (a) #output = 8000
print (d) #output = {'name': 'python',
                     'duration': '2 months'}
```

→ Dictionary functions :-

- get()
- keys()
- values()
- items()
- del()
- pop()
- dict()
- update()
- clear()

→ Create a file name :- dict4.py

```
d = {'course': 'Python',
     'fees': 8000,
     'duration': '2 months'}
c = d.get('course')
print(c) #output = Python
```

```
c1 = d['course']
print(c1) #~~print~~ output = python

for a in d.keys():
    print(a) #output = course
                       fees
                       duration

for b in d.values():
    print(b) # output = python
                        8000
                        2months

~~for d e in d.~~
for e,f in d.items():
    print(e,f) #output = course python
                         fees 8000
                         duration 2months

# del d['fees']
# print(d) #output = {'course':'python',
                       'duration':'2months'}
d.pop('duration')
print(d) #output = {'course':'python',
                    'fees':'8000'}
print(d.pop('duration')) #output = 2months
```

→ Create a filename: dict5.py

```
d = dict(name = 'python', fees = 8000)
print(d) #output = {'name':'python',
                    'fees':8000}
```

→ Carry on in dict6.py

```
d={'name':'python',
   'fees':8000,
   'duration':'2months'}
d.update({'fees':10000})
print(d) #output={'name':'python',
                  'fees':10000,
                  'duration':2 months}
d['desc']="This is python"
print(d) #output={'name':'python',
                  'fees':10000,
                  'duration':2 months,
                  'desc':This is python}
d["course"]="python"
d["name"]="Ramesh"
print(d) #output={'name':'Ramesh',
                  'fees':10000,
                  'duration':2months,
                  'desc':This is python}
```

▲ Python Nested Dictionary
(Dictionary under dictionary)

⇒ Nested Dictionary means putting a dictionary inside another dictionary.

⇒ It's a collection of dictionaries into one single dictionary.

→ Create a file name :- ~~nes~~ nested_dict.py

```
Course = { 'php': {'duration': '2 months', 'fees': '15000'},
           'python': {'duration': '2 months', 'fees': '16000'},
           'Java': {'duration': '2 months', 'fees': '17000'}}

print(course)
print(course['php']) #output = {'duration':
                                '2 months', 'fees': 15000}
print(course['php']['fees'])
                                #output = 15000
for k, v in course.items():
    print(k, v) #output =

    print(k, v['duration'], v['fees'])
```

▲ Tuple in Python

⇒ It is ordered data type
→ work /denoted by ( )
→ It is inmutable ⇒ Filename :- tuple.py

```
→ t = (20, 30, 40, 50)
        0   1   2   3
q = t[2]
print(q)  #output = 40
print(type(t)) #output = <class - tuple>
r = ~~(40)~~ ("python")
print(type(r)) #output = <class - str>
```

```
t = (20, 30, 40, 50)
     0   1   2   3

l = len(t)
print(l)  #output = 4
For a in range(l):
    print(a)  ~~#output =~~
    print(t[a]  #output = 20
                          30
                          40
                          50.
```

<u>Alternator method</u>

```
for a in t:
    print(a)   #output = 20
                         30
                         40
                         50
```

→ Function of tuples:

(1) min()
(2) max()
(3) count()
(4) index()
(6) sum()

→ Create a file name: tuple1.py

```
t = (20, 30, 40, 50)
     0   1   2   3

r = len(t)
print(r)  #output = 4
```

```
m = min(t)
print(m) #output = 20
h = max(t)
print(h) #output = 50
o = t.count(20)
print(o) #output = 1
i = t.index(40)
print(i) #output = 2.
s = sum(t)
print(s) #output = 140
s = sum(t,10)
print(s) #output = 150
```

Practice of tuple

- Tuple :-

A tuple is a collection which is ordered and inmutable iterating through tuple is faster than with list.

Create a file name :- tup.py

```
t = (10,20,30,40,60)
```

Accessing Elements in a Tuple

```
n = t[0]
print(n) #output = 10
print(type(t)) #output = <class-tuple>
```

- Iterating Through a Tuple :-

Using a for loop we can iterate through each item in a tuple.

• Tuple function:—

```
t = (10, 20, 30, 40, 50)
l = len(t)
print (l) # 5 = output
for a in range(l):
    print (a) #output = 0
                        1
                        2
                        3
                        4
print (t[a]) #output = 10
                       20
                       30
                       40
                       50
```

<u>Next method:</u>

```
for b in t:
    print (b) #output = 10
                        20
                        30
                        40
                        50
```

- min()
- max()
- count()
- index()
- sum()

```
m = min(t)
print(m) #output = 10
```

```
n = max(t)
print(n)  #output = 50
o = t.count(10)
print(o)  #output = 1
r = t.index(50)  ~~#output = 4~~
print(r)  #output = 4
s = sum(t)
print(s)  #output = 150
print(s, 10)  #output = 160
print(s, 30)  #output = 180
```

Sets in Python

- → It is unordered
- → It is unindex
- → represent by set() or { }
- → It has only values / not a keys
- → It does not take ~~rep~~ repeated value.
- → It use-unique type of value.
- → create a filename:- sets.py

```
s = {10, 20, 30}
for a in s:
    print(a)  #output = 10
                        20
                        30
```

→ sets function :-

- set()
- add()
- pop()
- remove()
- clear()
- discard()
- update()

```
l = [10, 20, 30]  → This is a list data type.
s = set(l)
print(s) #output = {10, 20, 30}
 s.add(40)
 print(s) #output = {10, 20, 30, 40}
# s.pop()
# print(s) #output = {10, 20, 40}
# print(s.pop()) #output = 30
# s.remove(20)
# print(s) #output = {10, 30, 40}
# s.discard(20)
# print(s) #output = {10, 30, 40}
# s = {10, 20, 30, 40}
# s.clean() #output = set()
#  print(s)
 l = [30, 40, 60]
s.update(l)
print(s) #output = {10, 30, 30, 40, 60}
```

Practicle of sets

- Sets :-
  A set is a collection which is unordered and unindexed, that is iterable, mutable, and has no duplicate elements. In python sets are written with curly brackets.

- Creat a set :- Filename = set1.py

```
s = {10, 20, 30, 40}
print(s) #output = {10, 20, 30, 40}
```

- Iterating Through a set

```
s = {10, 20, 30, 40}
for a in s:
    print(a) #output = 10
                       20
                       30
                       40
```

→ sets functions :-

(1) set()

(2) remove()

Note :- If the item to remove does not exist, remove() will raise an error.

(3) discard()

Note :- If the item to remove does not exist, discard() will NOT raise an error.

(4) pop()
(5) clear()
(6) add()
(7) update()

Create a filename:- sets4.py

```
l = [10,20,30,40]
s = sets()
print(s) #output = {10,20,30,40}
# s.remove(10)
# print(s) #output = {30,20,40}
# s.discard(40)
#print(s) #output = {10,30,20}
# s.pop()
# print(s) #output = {10,30,20}
# print(s.pop()) #output = 40
# s.clear() #output = set()
# s.add(60) #output = {10,20,30,40,60}
~~# s.update~~ l = [10,80,90]
~~print~~ s.update(l)
print(s) #output = {10,20,30,40,
                    60,80,90}
```

3rd Note

Python Full Beginned to Advanee.

wscubetech - YT.

▲ Functions in Python

* User Define function

→ • Simple function
• Function with arguments
• Return type

• simple function :- Filename :- simplefunction.py

Defined
```
def simplefunction():
    print("welcome to coscubetech")
```

call
```
simplefunction() #output = welcome to
                         coscubetech
```

```
def sumdata(a, b):
    print(a+b)
a = 10
b = 20
sumdata(a, b):
    ~~print(a+b)~~ #output = 30
~~sum~~ m1 = 10
    m2 = 50
sumdata(m1, m2): #output = 60
    print(m1+m2)
```

⇒ A function is a block of statements which can be used repetitively in a program. It saves the time of a developer. In Python concept of function is same as in other languages.

→ You can pass data, known as parameters, into a function.

- Creating a Function

→ In python a function is defined using the def keyword:

- Creating a function :- filename :- simplefun.py

```
def showdata():
    print("welcome to coscubetech")
```

- Calling function

```
showdata()
output = welcome to coscubetech
```

→ create a filename :- simplefun1.py

```
def showdata():
    print("welcome to coscubetech")
showdata() #output = welcome to coscubetech
showdata() #output = welcome to coscubetech
```

- Function Arguments – ~~Filename :- Argu.py~~

→ information can be passed to functions as parameter

```
def sum(a,b):
    print(a+b)
```

Calling with Arguments

```
sum(20,10)
output = 30
```

```
def sum (a,b):
    print(a+b)
sum (20,10) #output = 10
sum(30,40) #output = 70
sum (30,30) #output = 60
```

- Default parameter value ~~Filename: para.py~~

```
def sum (a,b=1):
    print(a+b)
```

Calling with Arguments

```
sum (20) #output = 21
sum (20,10) #output = 30
```

```
def sum(a,b=2):
    print(a+b)
sum (20) #output = 22
sum(40,20) #output=60
```

- Return Values :-

To let a function return a value, use the return statement:

```
def square(x):
    return x*x, x**2
s = square(5) ~~#output = 25~~
print(s) #output = (25 ,10)
```

▲ Modules in Python

* How to create Your own Module in python

Filename:- module1.py ~~checking.py~~	checking.py ~~module1.py~~

def sum(a,b):
c=a+b
return c
def mul(a,b):
c=a*b
return c
	import module1
	print(module1.sum(10,20))
	#output=30
	
	Print(module1.mul(10,20))
	#output=200
	
	2nd method of import
	
	from module1 import sum
	print(sum(10,30))
	#output=40
	
	from module1 import *
	print(mul(10,20))
	#output=200
	print(sum(30,40))
	#output=70

▲ Math Module in Python

⇒ Math functions in Python.

⇒ we have to import math inbuilt function

~~* Math.~~

* MATH.CEIL(X) :-

Return the ceiling of x, the smallest integer greater than or equal to x. If x is not a float, delegates to x.__ceil__(), which should return an integral value.

```
x = 10.5
print(math.ceil(x)) #output = 11
```

→ Create a file name :- math.ceil.py

```
import math / from math import *
x = 10.2
print(math.ceil(x)) #output = 11
y = 10
print(math.ceil(y)) #output = 10
z = 12.2
print(math.ceil(z)) #output = 13
```

* MATH.Fabs(x)

⇒ Return the absolute value of x.

```
x = -10
print(math.fabs(x)) #output = 10
```

→ Create a file name :- math.fabs.py

```
import math / from math import *
x = -10
y = -11
z = 12
print(math.fabs(x)) #output = 10.0
print(math.fabs(y)) #output = 11.0
print(math.fabs(z)) #output = 12.0
```

* MATH.FACTORIAL (x)
=> Return x factorial as an integer. Raises value 'error' if 'x' is not integral or is negative

```
x = 5
print(math.factorial(x)) #output = 120
```

→ Create a filename:- math_factorial.py

```
import math / from math import*
x = 5
print(math.factorial(x)) #output = 120
y = 6
print(math.factorial(y)) #output = 720
z = 8
print(math.factorial(z)) #output = 40320
```

* Math.floor(x)
=> Return the floor of x, the largest integer less than or equal to x. If x is not a float, delegates to x.__floor__(), which should return an integral value.

```
x = 10.5
print(math.floor(x) #output = 10
```

→ Create a filename:- math.floor.py

```
x = 11.2
print(math.floor(x)) #output = 11
y = 13.4
print(math.floor(y)) #output = 13
z = 14.0
print(math.floor(z)) #output = 14
```

~~http://~~ https://docs.python.org/3.10/library/math.html
⇒ For more function self study

* math.fsum(iterable)
⇒ Return an accurate floating point sum of values in the iterable.
→ All value must be integer to use this function.
→ Create a file name:- math.fsum.py

```
import math / from math import*
l = [10,20,30,40]
print(math.fsum(l)) # output = 100.0
m = [10,20,30,40,100,200]
print(math.fsum(m)) #output = 400.0
```

* MATH.SQRT(x)
⇒ Return the square root of x.

→ Create a file name:- math.sqrt.py

```
import math / from math import*
x = 16
print(math.sqrt(16)) # output = 4.0
z = 64
print(math.sqrt(z)) #output = 8.0
y = 9
print(math.sqrt(y)) # output = 3.0
a = 5
print(math.sqrt(a)) # output = 2.2360679774997 9
b = 81
print(math.sqrt(b)) #output = 9.0
```

Random Module in Python

⇒ It is inbuit module in Python.

* Python Random randint() Method
  → Return a number between 5 and 10 (both included):

```
import random
print(random.randrange(3,9))
```

* Python Random randrange() method
  → Returns a number between 3 (included) and 9 (not included)

```
import random
print(random.randrange(3,9))
```

* Python Random choice() method.
  → Return a random element from a list:

~~l = ["apple", "banana", "cherry"]~~

```
import random
l = ["apple", "banana", "cherry"]
print(random.choice(l))
```

⇒ Create a filename:- random_module.py

```
import random
n = random.randint(2,8)
print(n)
n1 = random.randrange(2,4)
print(n1)
```

```
l=[10,20,30,40]
n2=random.choice(l)
print(n2)
```

* random() — Returns a random float number between 0 and 1.
* shuffle() — Takes a sequence and returns the sequence in a random order.
* uniform() — Returns a random float number between two given parameters.

→ Create a filename :- random.py

```
import random
r=random.random() # float value '0' to '1'
print(r)
l=[10,20,30,40]
random.shuffle(l)
print(l)
u=random.uniform(3,9) #Give float value
print(u)
```

Datetime Module in Python — import datetime

⇒ import a module named datetime to work with dates as date objects.

```
import datetime
x=datetime.datetime.now()
Print(x)
```

The date contains year, month, day, hour, minute, second, and microsecond.

→ Creating Date objects

```
x=datetime.datetime(2021,1,18)
print(x)
```

⇒ Create a filename:- datetime.py

```
import datetime
x = datetime.datetime.now()
print(x) #output = current date and time
y= datetime.datetime(2022,2,22)
print(y) #output = 2022-02-22 00:00:00
```

* Python Dates

→ he method is called strftime(), and takes one parameter, format, to specify the format of the returned string:

%b – Month name, short version – Dec
%B – Month name, full version – December
%m – Month as a number – (01-12) – 12
%y – Year, short version without century 18
%Y – Year, full version 2018
%H – Hour 00-23 17
%I – Hour 00-12 05
%p – AM/PM PM
%M – minute 00-59 41
%S – second

→ Create a filename:- python_dates.py

```
import datetime
now = datetime.datetime.now()() # Current date and
                                  time.
year = now.strftime("%Y")
print ("year:", year)
x = datetime.datetime.now()
m = x.strftime("%Y")
print(x)
print(m)
n = x.strftime("%y")
print(n)
o = x.strftime("%b")
print(o)
p = x.strftime("%B")
print(p)
q = x.strftime("%m")
print(q)
r = x.strftime("%H")
print(r)
s = x.strftime("%I")
print(s)
t = x.strftime("%p")
print(t)
u = x.strftime("%M")
print(u)
v = x.strftime("%S")
print(v)
```

Project 1

Random Number Guessing Game

[Using Random Module]

=> Create a file name :- Project1.py

~~import random~~
~~Cnumber = random.randrange(1, 101)~~
~~user input = int(input("Enter your Number :- "))~~
~~if usersinput > cnumber:~~
~~print("your guess number is high")~~
~~elif cnumber > user input:~~
~~print("your guess number is too low")~~

~~else:~~
~~print("your guess number is equal")~~

```
import random
cnumber = random.randrange(1,101)
usersinput = int(input("Enter your number :- "))
If usersinput > cnumber:
    print("your guess number is high")
    print("Computer number", cnumber)
    print("your guess number is high")
elif usersinput < cnumber:
    print("Computer number", cnumber)
    print("your guess number is too low")
else:
    print("Computer number", cnumber)
    print("your guess number is equal to
        computer number")
```

Project-2

▲ Rock, Paper, Scissors Game (Using Random Module)

⇒ Wining Rules as follows:

Rock vs paper → paper wins
Rock vs scissor → Rock wins
Paper vs scissor → scissor wins

→ Create a filename :- rsp1.py

```
import random
l=["rock","scissor","paper"]
'''
rock vs paper -> paper wins
rock vs scissor -> Rock wins
Paper vs scissor -> scissor wins
'''
while True:
    ccount=0
    ucount=0
    uc=int(input('''
Game Start.....
1 Yes
2 No/Exit
''')))
    if uc==1:
        for a in range(1,6):
```

```
usersinput=int(input('''
1 Rock
2 Scissor
3 paper
''''))
if usersinput==1:
    uchoice = "rock"
elif usersinput==2:
    uchoice="scissor"
elif usersinput==3:
    uchoice = "paper"
cchoice=random.choice(l)
if cchoice ==uchoice:
    print("Computer value",cchoice)
    print("user value", uchoice)
    print("Game Draw")
    ucount = ucount +1
    ccount = ccount+1
elif (uchoice=="rock" and cchoice==
 "scissor") or (uchoice=="paper" and
 cchoice=="rock") or (uchoice=="scissor"
 and cchoice=="paper"):
    print("computer value", cchoice)
    print("user value", uchoice)
    print("you win")
    ucount =ucount+1
else:
    print("computer value",cchoice)
    print("user value",uchoice)
```

```
        print("computed coin")
        ccount=ccount+1

if ucount==ccount:
    print("Final Game Draw---")
    print("user score", ucount)
    print("computer score", ccount)
elif ucount>ccount:
    print("Final you win the Game")
    print("user score", ucount)
    print("computer score", ccount)
else:
    print("Final Computer win the Game")
    print("user score", ucount)
    print("computer score", ccount)
else:
    break
```

▲ Pickle Module

⇒ The pickle module implements a fundamental, but powerful algorithm for serializing and de-serializing a python object structure.

* Storing data types with pickle

→ You can pickle objects with the following data types:

- Booleans,
- Integers,
- Floats,
- Complex numbers,

- (normal and unicode) strings,
- Tuples,
- Lists
- Sets, and
- Dictionaries

* Pickling files

=> To use pickle, start by importing it in Python.

import pickle

=> pickle has two main methods. The first one is dump, which dumps an object to a file object and the second one is load, which loads an object from a file object.

* Python pickle functions
  - dump() - This function is called to serialize an object hieardhy.
  - load() - This function is called to de-serialize a data stream.

- Pickling with dump()

=> Pickling data is done via the dump() function. It accepts data and a file object. The dump() function then serializes the data and writes it to the file. The syntax of dump() is as follows:

syntax: dump(obj, file)

Argument	Description
obj	object to be picked
file	file object where pickled data will be written.

→ create a file name :- pickle.py and writedata.txt

```
import pickle
example = {1:"6", 2:"2", 3:"8"}
pickle_write = open("text.txt","wb")
pickle.dump(example, pickle_write)
pickle_write.close()
```

→ import pickle

```
l = [10, 20, 30, 40]
file = open("writedata.txt","wb")
pickle.dump(l, file)
file.close()
```

- Unpicking with load()

⇒ The load() function takes a file object, reconstruct the objects from the pickled representation, and return it.

Its syntax is as follows:

```
import pickle
f = open("text.txt","rb")
pickle.load(f)
```

Argument	Description
file	File object from where the Serialized data will be read

→ ~~import pickle~~ filename :- pic.py

```
import pickle
file = open("writedata.txt", "rb")
l = pickle.load(file)
print(l) #output = [10, 20, 30, 40]
```

▲ JSON in Python

⇒ JSON (JavaScript object Notation).

⇒ It is mainly used for storing and the transferring data between the browser and the server.

⇒ JSON is text, written with JavaScript object notation.

⇒ Python too supports JSON with a built in package called Json.

<u>import Json</u>

* JSON supports mainly 6-data types:-
  1. String
  2. number
  3. boolean
  4. null
  5. object
  6. array

In Python, JSON exists as a string. For e.g.

p = '{"name": "ws", "lang": ["Python", "Java"]}'

* Converting Python objects to JSON

```
import json
blog={"URL": "www.wsecubetech.com", "name": "wsecubetech"}
to_json = json.dumps(blog)
```

⇒ Create a filename :- j.py

```
import json
d={ 'course_name': 'Python',
    'fees': 15000
    }
f = json.dump(d)
print(f) #output =
print(type(f)) #output = class <str>
```

▲ Convert JSON Data to Python object

⇒ If you have a JSON string, you can parse it by using the json.loads() method.

```
import json
#some JSON:
x='{"cname": "Python", "fees": 1200, "duration":
    "2 months"},
#Parse x:
y=json.loads(x)
# the result is a python dictionary:
```

→ Create a filename :- Json.j.py

```
import Json
d='{"cname":"Python","fees":12000,
    "duration":"2 months"}'
x=Json.loads(d)
print(type(x)) #output=class <dict>
print(x)
for a in x:
    print(a,x[a])
```

```
l='[{"cname":"Python","fees":12000,
    "duration":"2months"}]'
y=Json.loads(l)
print(type(y)) #output=class <list>
print(y)
for a in y:
    print(a)
```

▲ Reading and writting JSON file in python

⇒ Write or take one Json file to your local file and rename it as posts.Json

→ create filename :- read.Json.py

```
import Json
file=open("posts.Json","r")
x=file.read()
finaldata=Json.loads(x)
print(finaldata)
```

```
print(type(finaldata))
for a in finaldata:
    print(a['title'],a['usedId'])
```

▲ Object Oriented Programming (OOPs) (class, object)

⇒ Create a filename:- oops.py

```
~~class Democlass:~~
    ~~a = 10~~

~~demoobject = DemoClass()~~
~~print(demoobject.a) #output = 10~~
~~demoobject1 = DemoClass()~~
~~print(demoobject1.a) #output = 10~~
```

```
class Democlass:
    a = 10
    def sumvalue(self):
        print(20+30)
demoobject = DemoClass()
print(demoobject.a) #output = 10
demoobject1 = Democlass()
print(demoobject1.a) #output = 10
demoobject.sumvalue(); #output = 50
```

▲ Object Oriented Programming (Method, Constructor)

⇒ Create a file name: method_cons.py

```
class DemoClass:
    a = 10
    def __init__(self):
        print("welcome to wscubetech")
    def showvalue(self):
        self.c = self.a * self.a
        print(self.c)
    def showvalue1(self,a,b):
        print(a+b)
obj = DemoClass()
obj.showvalue()  # output = 100
obj.showvalue1(20,30)  # output = 50
```

▲ Inheritance in Python

⇒ Create a filename: inheritance.py

Double inheritance

```
class A:
    def displayA(self):
        print("welcome to wscubtech A")
class B(A):
    def displayB(self):
        print("welcome to wscubtech B")
```

```
class C(B):
    def displayB(self):
        print("Welcome to wscubetech c")

obj = C()
obj.displayA()
obj.displayB()
obj.displayC()
```

-> Create a file name : inheritance1.py

multiple inheritance

```
class A:
    def displayA(self):
        print("welcome to wscubeteech A")
class B:
    def displayB(self):
        print("welcome to wscubeteech B")
class C(A,B):
    def displayC(self):
        print("welcome to wscubeteech c")

obj=C()
obj.displayA()
obj.displayB()
obj.displayC()
```

Encapsulation in python (Getter and setter)
(object Oriented Programming)

=> Create a filename :- encapsulation.py

```
class student:
    def __init__(self):
        self.__name = " "
    def getname(self):
        return self.__name
    def setname(self, nm):
        self.__name = nm
obj = student()
obj.setname("Testing")
name = obj.getname()
print(name)
```

* Encapsulation

=> An object variables should not always be directly accessible.

=> The methods can ensure the correct values are set. If an incorrect value is set, the method can return an error.

=> Create a filename: encapsulation2.py

```
Class student:
    __name = "Ravi"
```

```
def __displayinfo(self):
    print("welcome to coscubeteeh")
def __init__(self):
    print(self.__name)
    self.__displayinfo()

obj=student()
```

▲ Polymorphism (object Oriented Programming)

⇒ Polymorphism means same function name (but different signatures) being uses for different types.

⇒ Create a filename: polymorphism.py

```
#l=[10,20,30,40]
#print(len(l))
#s="welcome"
#print(len(s))

class Ws:
    def displayinfo(self,name=' '):
        print("welcome to coscubeteeh"+name)
obj=Ws()
obj.displayinfo()
obj.displayinfo('python')
```

→ Create a filename :- polymorphism1.py

```
class Ws:
    def displayinfo(self):
        print("welcome to Wscubetech")

class IIP(Ws):
    def displayinfo(self):
        print("welcome to IIP")
obj = IIP()
obj.displayinfo()
```

To display both result : (we use super().fun

```
class wv:
    def displayinfo(self):
        print("welcome to wv")
class IV(wv):
    def displayinfo(self):
        ~~print("welcome to IV")~~
        super().displayinfo()
        print("welcome to IV")

obj = IV()
obj.displayinfo()
```

▲ Method Overloading and method overriding in python.

* Method overloading

⇒ • Method overloading is one concept of polymorphism.
- It comes under the elements of oops.
- It is worked in the same method names and different arguments
- Arguments different will be based on a number of arguments and types of Arguments.

→ Create a filename :- overloading.py.

```
class Area:
    def find_area(self,a=None,b=None):
        if a!=None and b!=None:
            print("Area of rectangle:",(a*b))
        elif a!=None:
            print("Area of square:",(a*a))
        else:
            print("Nothing to find")

obj1 = Area()
obj1.find_area() #output = Nothing to find
obj1.find_area(10) #output= 100
obj1.find_area(10,20) #output= 200
```

* Method Overriding

=> • Method overriding is the method having the same name with the same arguments.
• It is implemented with inheritance also.
• It mostly used for memory reducing processes.

→ Create a filename : overriding.py

```
class A:
    def showData(self):
        print("I am in A class")
class B(A):
    def showData(self):
        print("I am in B class")

obj = B()
obj.showData() #output = I am in B
                         class
```

4th Note

Python Full Beginner to Advance

wscubetech - YT

▲ Bike Rental System → using OOPs

⇒ (1) Display available bikes
(2) Request a bike for rent (100Rs → 1 qty)
(3) Exit

⇒ Create a filename: bike_System_Rental.py

```
class bikeshop:
    def __init__(self,stock):
        self.stock = stock
    def displayBike(self):
        print("Total Bikes", self.stock
    def rentfor Bike(self,q):
        if q<=0:
            print("Enter the positive value or
                greater than zero")
        elif q>self.stock:
            print("Enter the value (less than
                stock)")
        else: self.stock = self.stock - q
            print("Total prices", q*100)
            print("Total Bikes",self.stock)

while True:
    obj = bikeshop(100)
    uc = int(input("""1. Display Stocks
                    2. Rent for Bike
                    3. Exit : """))
```

```
if uc==1:
    obj.displayBike()
elif uc==2:
    n=int(input("Enter the Qty:-"))
    obj.rentForBike(n)
else:
    break
```

Python Errors and Built-in Exceptions

- Python Errors and Built-in Exceptions

⇒ These errors can be broadly classified into two classes:

- Syntex errors
- Logical errors (Exceptions)

* Python Syntex errors — Filename.syn.py

```
a=10
b=20
if a==b
    print("No")
else
    print("yes")
#output= Syntex error
```

We can handle Syntex errors

⇒
```
a=10
b=30
if a==b
    print("yes")
else
    print("No")
```
#output= Syntex error.

* Python Logical Errors (Exceptions)

⇒ print(1/10)
~~zivi~~ zero division Error : divisi
-on by zero

We can handle the exceptions error

l = [10, 20, 30, 40]
print(l[6]) ~~ad~~ Index error

⇒ eception.py ⇒ Filename created

print (a/0) #output = zero division error
l = [10, 20, 30]
print (l[4]) #output = index error : list index out of range.

▲ Python Exception Handling

⇒

(1) Zero Division Error
⇒ e.g. a = 10
print (a/0)

(2) Name Error
⇒ print(b)

(3) Type Error
⇒ print ('10' + 10)

(4) Value Error

⇒ a = int(input("Enter the value"))

a = "hello" ✗

↓

ValueError

a = 10 ✓

(5) Index Error

⇒ l = [1, 2, 3, 4]

0 1 2 3

print(l[4])

(6) Key Error

⇒ d = {'name': 'Testing', 'fees': 8000}

print(d['fee'])

print(d['fees'])

(7) Modul Not Found Error

⇒

used module	created module
import cals	cal.py

(8) Import Error

⇒

Used module	Created module
from cal import sum1	cal.py def sum(): print(10+20)

* Types of Exceptions:
  (1) Overflow Error
  (2) Zero Division Error
  (3) Floating Point Error
  (4) Key Error
  (5) Index Error
  (6) File Not Found Error
  (7) Class Not Found Error
  (8) Memory Error

* Component of Exception Handling

⇒ try → error prone code will be under try block

except → to handle the exception

else → if there is no exception in code

finally → it runs in both condition

raise → Exception through purpose we use it

⇒ Create a filename:- exception_handling.py

```
#Exception
a=input("Enter the number")
x = a/5
print(x)  #output= TypeError
```

```
#Handling
    a=input("Enter a number:")
try:
    x=a/5
except TypeError:
    print("Invalid Input Type")
else:
    print(x)
 #output = Invalid Input Type
 ~~#output = Invalid Input Type~~
finally:
    print("Thank You")
```

optional / Not necessary to write

⇒ Create a file name: zeroDivisionError.py

```
#Exception / Error
a=int(input("Enter the number:"))
b=int(input("Enter the number2:"))
x=a/b
print(x) #output=zeroDivisionError.

#Handling
a= ~~input~~ int(input("Enter the number:"))
b= int(input("Enter the number2:")) click=0
try:
    x=a/b
```

```
except ZeroDivisionError:
    print("Don't input zero")
else:
    print(x)
finally:
    print("Thank You")
```

Database

Python - SQLite (DB create and connection)

*SQLite in Python (create connect & table)

⇒ create a filename:- sqliteconnet.py

```
import sqlite3
conn=sqlite3.connect("sqlite.db")
conn.execute('''
        create table student(
            st_id INT AUTO_INCREMENT PRIMARY
                                    KEY,

            st_name VARCHAR(50),
            st_class VARCHAR(10),
            st_email VARCHAR(30)
                                    )''')

conn.close()
```

* Python = Installing DB Browser for SQLite

⇒ Install : DB Browser for SQLite

* Insert Query - SQlite

⇒ create filename:- insert.py

```
import sqlite3
conn = sqlite3.connect("sqlite.db")
ins = '''
        insert into student (st_name,
        st_class, st_email) VALUES ('RAM',
    '12th', "ram@gmail.com") '''
conn.execute(ins)
conn.commit()
conn.close()
```

```
conn = sqlite3.connect("sqlite.db")
inse = '''
        insert into student (st_name, st_class,
        st_email) values ("Ravi", "12th",
    "ravi@gmail.com") '''
conn.excute(inse)
conn.commit()
conn.close()
```

* Select Query, Order By and Limit

⇒ Create a filename:- selectdata.py

```
import sqlite3
# conn = sqlite3.connect("sqlite.db")
# data = conn. ~~execut~~ execute("SELECT * FROM student")
# for n in data:
#         print(n)

# print("STUDENT ID", "STUDENT NAME", "STUDENT CLASS",
#         "STUDENT EMAIL")
# for n in data:
#         print(n)
# for n in data:
#         print(n[0], n[1], n[2], n[3])


conn=sqlite3.connect("sqlite.db")
data1 = conn.execute("SELECT * FROM student
        limit 0,2 ")
for n in data:
    print(n)
```

~~SQLite Delete Query~~

~~* SQL~~

* Delete query in sqlite

=> Create a filename:-delete.py

```
import sqlite3
conn = sqlite3.connect("sqlite.db")
st_id = input("Enter the student name:-")
conn.execute("DELETE FROM student where
                st_id = "+st_id)
conn.commit()
conn.Close()
```

* SQLite Update Query

=> Create a filename:- update.py

```
import sqlite3
conn = sqlite3.connect("sqlite.db")
#conn.execute("""
                update student set st_name =
                'ram 1' where st_email =
                "ram@gmail.com" """)
#conn.commit()
#conn.Close()
```

```
conn.execute("""update student set st_id =
                '1', st_class = "10th" where
                st_email = "ram@gmail.com")
conn.commit()
conn.close()
```

*SQLite WHERE clause
(Searching Query)

⇒ Create a file name:- searching.py

```
import sqlite3
conn=sqlite3.connect("sqlite.db")
data=conn.execute("SELECT* FROM student")
for n in data:
    prin(n)
Print(" ")
#Searching result from data

# data=conn.execute("SELECT* FROM student
#          where st_name='shyam'")
# for n in data:
#     print(n)

st_name=input("Enter the student Name:-")
st_class=input("Enter the student class:-")
# data=conn.execute("SELECT* FROM student where
        st_name = '"+st_name+"' ")
#for n in data:
        print(n)
data = conn.execute("SELECT* FROM student where
        st_name like '%"+st_name+"%' and st_class
        = '"+st_class"' ")
for n in data:
        print(n)
```

* Python - SQLite - Join

⇒

	st_id	st_name	st_class	st_email
	Filter	Filter	Filter	Filter
1	1	Ram	12th	abc@gmail.com
2	2	Raj	12th	raj@gmail.com
3	3	Ws	12th	ws@gmail.com

	ees_id	st_id	fees_amount
	Filter	Filter	Filter
1	1	1	2000
2	2	1	3000
3	3	2	3000
4	4	3	3000
5	5	1	3000
6	6	2	3000
7	7	2	5000

- (INNER) JOIN :

  Returns records that have matching values in both tables

- LEFT (OUTER) JOIN:

  Returns all records from the left table, and the matched records from the right table

- RIGHT (OUTER) JOIN:

  Returns all records from the right table, and the matched records from the left table.

- FULL (OUTER) JOIN:

Returns all records when there is a match in either left or right table

Note:- sqlite3.OperationalError: RIGHT and FULL (OUTER) JOINs are not currently supported. ⇒ on developing

⇒ Create a filename: sqlite_Join.py

```
import sqlite3
conn=sqlite3.connect("sqlite.db")
print("Student ID","Student Name","Student Fees")
#data=conn.execute("SELECT * FROM fees as f
        inner Join student as s on f.st_id=s.st_id")
#for a in data:
#       print(a)
#conn.close()
data = conn.execute("SELECT f.st_id,s.st_name,
        f.fees_amount FROM fees as f inner
        Join student as s on f.st_id=s.st_id")
for a in data:
    print(a)
conn.close()
```

MySQL Database

▲ Install XAMPP on Windows - PyMySQL

* Python MySQL pymysql
  → pip install ~~pymysql~~, then
  → Xampp Installation
    ⇒ localhost/phpmyadmin/
  → start apache & MySQL

* Python : Creating MySQL Database and connection.

⇒ Create a filename:- database.py

```
import pymysql as mq
~~myobj = mq.connect(#~~
#ServerName -> localhost
#Username -> root
#Password -> Blank
myobj = mq.connect("localhost","root","")
cursor obj = myobj.cursor()
try:
    db = "create database school"
    cursorobj.execute(db)
    print("database created")
except:
    print("database error..")
```

→ run the file from terminal

* MySQL Data Types

⇒ Types of data-types:-

- INT → length(11) → store number
- BIGINT → length(20)
- TEXT → length(0-60000) — store string
- VARCHAR → length (0-255)
- LONGTEXT → length (<60000)
- DATE → YYYY-MM-DD
- DATETIME → YYYY-MM-DD HH:MM:SS
- TIMESTAME → store present time on data
- TINYINT → length (3)
- TIME → HH: MM: SS
- YEAR → YYYY

* Python MySQL Create Table

⇒ open localhost/phpmyadmin/

⇒ Create a filename: create.py

```
import pymysql as mq
conn=mq.connect(
        host="localhost",
        user="root",
        passwd="",
        database="school")
```

~~mysqlc = conn.cursor()~~

```
mysqlc = conn.cursor()
# Table student(st_id, st_name, st_class,
                st_email)
tc = " create table student (st_id
    INT primary key auto_increment,
    st_name Varchar(50), st_class
    varchar(10), st_email varchar(50))"

mysqlc.execute(tc)
```

* MySQL Insert Query using Python:

⇒ Create filename: insert1.py

```
import pymysql as mq
mysql = mq.connect("localhost", "root", " ",
        "school")
mycursor = mysql.cursor()

try:
    # st_id, st_name, st_class, st_email
    ins = "insert into student(st_name,
        st_class, st_email) values(%s,
    %s, %s)"
    t = ("ram", "12th", "ram@gmail.com")
    mycursor.execute(ins, t)
    mysql.commit();
    print("Insert Data")
except:
    print("Error")
```

⇒ Run

Next Insert multiple entry

```
import pymysql as mq
mysql = mq.connect("localhost","root","","school")
mycursor = mysql.cursor()
try:
    #st-id, st-name, st-class, st-email
    ins = "insert into student (st-name, st-class, st-email) values (%s,%s,%s)"
    ~~t = ("ravi", "12th", "ravi@gmail.com")~~
    ~~mycursor.execute(ins,t)~~
    ~~mysql.commit();~~
    ~~print("Insert Data")~~
~~except:~~
    ~~print("Error")~~
    t = [("raj", "10th", "raj@gmail.com"),
         ("ravi", "12th", "ravi@gmail.com"),
         ("testing", "10th", "testing@gmail.com")]
    mycursor.executemany(ins,t)
    mysql.commit();
    print("Insert Data")
except:
    print("Error---")
```

* MySQL Select Query using Python

⇒ Create a filename: select.sql.py

```
import pymysql as mq
mysql = mq.connect("localhost", "root", " ",
                   "school")
mycursor = mysql.cursor()

mycursor.execute("SELECT * FROM student")
print("{:<20} {:<30} {:<15} {:<30}".format
                    ("student Id",
                     "student Name",
                     "student Email",
                     "student class"))

try:
    myresult = mycursor.fetchall()
    for x in myresult:
        print("{:<20} {:<30} {:<15} {:<30}"
          .format(x[0], x[1], x[2], x[3]))
```

used to character size

used index number of selected column

```
except:
    print("error...")
```

⇒ Create a filename: select query.py

```
import pymysql as mq
# server connection or database connection

mysql = mq.connect("localhost", "root", " ", "school")
mycursor = mysql.cursor()
print("{:<15} {:<20} {:<20} {:<10}".format
      ("student Id", "student Name", "student Email",
      "student class"))

try:
    sql = "select * from student"
    mycursor.execute(sql)
    ~~mycursor.~~
    sdata = mycursor.fetchall()
    for s in sdata:
        ~~print(s)~~
        print("{:<15} {:<20} {:<20} {:<10}"
            .format(s[0], s[1], s[2], s[3]))

except:
    print("Error..")
```

⇒ Run the program in terminal.

Selecting Columns ⇒ select Query

```
import pymysql as mq
mysql = mq.connect("localhost", "root", " ", "school")
mycursor = mysql.cursor()
mycursor.execute("select st_name, st_class from
                  student")
print("{:<30} {:<15}".format("student Name",
                  "student class"))

try:
    myresult = mycursor.fetchall()
    for x in myresult:
        print("{:<30}{:<15}".format(x[0], x[1]))

except:
    print("error")
```

Using The FETCHONE() Method

```
import pymysql as mq
mysql = mq.connect("localhost", "root", " ", "school")
mycursor = mysql.cursor()
mycursor.execute("select * from student")
print("{:<20} {:<30} {:<15} {:<30}".format
        ("student Id",
         "student Name",
         "student Email",
         "student class"))
```

```
try:
    i=mycursor.fetchone()
    print("{:<20}{:<30}{:<15}{:<30}".format
            (i[0],i[1],i[2],i[3])

except:
    print("error")
```

<u>Next</u>:

```
import pymysql as mq
mysql=mq.connect("localhost","root"," ","school")
mycursor=mysql.cursor()
print("{:<15}{:<20}".format("student Id",
            "student Name"))

try:
    sql="select st_id,st_name from student
        where st_id=2"
    mycursor.execute(sql)
    sdata=mycursor.fetchone()
    print("{:<15}{:<20}".format(sdata[0],
            sdata[1]))
except:
    print("Error")
```

*MySQL Delete Query using python

⇒ SQL has the DELETE command that is used to delete data from a table. The DELETE command delete a row in the table.

```
import pymysql as mq
mysql = mq.connect("localhost", "root", "",
                   "school")
```

Connection

mysql.commit() is required to make the changes, otherwise no changes are made to the table.

Delete single row

→ Create a filename: delete.sql.db.py

```
import pymysql as mq
mysql = mq.connect("localhost", "root", " ",
                   "school")
x = mysql.cursor()
id = input("Enter the student_id")
sql = "DELETE FROM student WHERE st_id = %s"
```

```
try:
    x.execute(sql,id)
    mysql.commit()
    print("Delete student Details Successfully.")

except:
    print("error..")
```

⇒

```
import pymysql as mq
mysql = mq.connect("localhost","root"," ","school")
cursorobj = mysql.cursor()
st-id = input("Enter The student Id")
sql = "DELETE FROM student where st-id = %s"
try:
    cursorobj.execute(sql,st-id)
    mysql.commit()
    print("student DELETED")
except:
    print("Error")
```

* MySQL Update Query using python

⇒

```
import pymysql as mq
mysql = mq.connect("localhost","root"," ",
        "school")
x = mysql.cursor()
```

```
st_id=input("Enter the student id")
sql="UPDATE student SET st_name=%s,
        st_class=%s,st_email=%s WHERE
        st_id=%s"
data=("Ram Singh","7th","ram123@gmail.com",
        st_id)
try:
    x.execute(sql,data)
    mysql.commit()
    print("Data updated.")

except:
    print("Error")
```

student

st-id	st-name	st_email	st_class
1	Ram	ram123@gmail.com	12th
2	Ravi	ravi123@gmail.com	12th

```
-> import pymysql as mq
mysql=mq.connect("localhost","root",
                 "","school")
cursorobj=mysql.cursor()
name=input("Enter the student name:")
class_name=input("Enter the class Name:")
st_id=input("Enter the student id:")
```

```
sql = "UPDATE student set st_name=%S, st_class
        =%S where st_id=%S."
data = (name, class_name, st_id)
try:
    cursorobj.execute(sql, data)
    mysql.commit()
    print("Data Updated")
except:
    print("Error")
```

* Python MySQL Order By and Limit

=> Create a filename: orderby.py

```
import pymysql as mq
mysql=mq.connect("localhost", "root", " ", "school")
mycursor=mysql.cursor()
print("{:<15} {:<20} {:<20}".format("student
  ID", "Student name", "student email", "student class"))
try:
    sql = "select * from student order by
        st_id DESC"
    mycursor.execute(sql)
    sdata = mycursor.fetchall()
    for a in sdata:
        print("{:<15} {:<20} {:<20} {:<20}".format
            (s[0], s[1], s[2], s[3]
except:
    print("Error")
```

→ ~~import mys~~

```
import pymysql as mq
mysql=mq.connect("localhost", "root", " ",
                 "school")

mycursor = mysql.cursor()
print("{:<15} {:<20} {:<20} {:<20}".format
      ("student Id", "student name", "student email"
      , "student class"))
try:
    sql="select * from student order by
         st_name ASC"
    mycursor.execute(sql)
    sdata = mycursor.fetchall()
    for s in data:
        print("{:<15} {:<20} {:<20} {:<20}"
              .format(s[0], s[1], s[2], s[3]))
except:
    print("Error")
```

→

```
import pymysql as mq
mysql = mq.connect("localhost", "root",
                   " ", "school")

mycursor = mysql.cursor()
print("{:<15} {:<20} {:<20} {:<20}".format
      ("student Id", "student name", "student
       email", "student class"))
try:
```

```
#ASC, DESC
sql = "select * from student order by
        st_id ASC LIMIT 0,2"
mycursor.execute(sql)
sdata = mycursor.fetchall()
for s in sdata:
        print("{:<15} {:<20} {:<20} {:<20}"
          .format(s[0], s[1], s[2], s[3]))
Except:
    print("Error")
```

* Python MySQL Searching and Filter Data

⇒ Create a filename: search.py

```
import pymysql as mq
mysql = mq.connect("localhost", "root", " ",
                   "school")
mycursor = mysql.cursor()
print("{:<15} {:<20} {:<20} {:<20}".format
      ("student Id","student Name", "student class",
       "student Email"))
try:
    name = input("Enter the student Name:-")
    sql = "select * from student where st_name =
            ('"+name+"')"
    mycursor.execute(sql)
    ~~sdata~~
```

```
sdata=mycursor.fetchall()
for s in sdata:
        print("{:<15} {:<20} {:<20}
          {:<20}".format(s[0],s[1],
          s[2],s[3]))
except:
        print("Error")
```

⇒

```
import pymysql as mq
mysql=mq.connect("localhost","root"," ",
                "school")
mycursor=mysql.cursor()
print("{:<15} {:<20} {:<20} {:<20}".format
    ("student Id","student Name","student class",
     "student Email"))
try:
    name=input("Enter the student Name:-")
    # = like
    sql="select * from student where st_name
        like '%"+name+"%'"
    mycursor.execute(sql)
    sdata=mycursor.fetchall()
    for s in sdata:
        print("{:<15} {:<20} {:<20} {:<20}"
            .format(s[0],s[1],s[2],s[3]))
except:
    print("Error")
```

=>

```
import pymysql as mq
mysql = mq.connect("localhost", "root", " ",
                   "school")
mycursor = mysql.cursor()
print("{:<15} {:<20} {:<20} {:<20}".format
    ("student Id", "student Name", "student
      class", "student email"))

try:
    name = input("Enter the student name:-")
    classname = input("Enter the student class
                      name:-")

    # = like
    sql = "select * from student where st_name
          like '%"+name+"%' and st_class='"
                +classname+"'"
    mycursor.execute(sql)
    sdata = mycursor.fetchall()
    for s in sdata:
        print("{:<15} {:<20} {:<20}
          {:<20}".format(s[0], s[1],[2],[3]
          ))
except:
    print("Error")
```

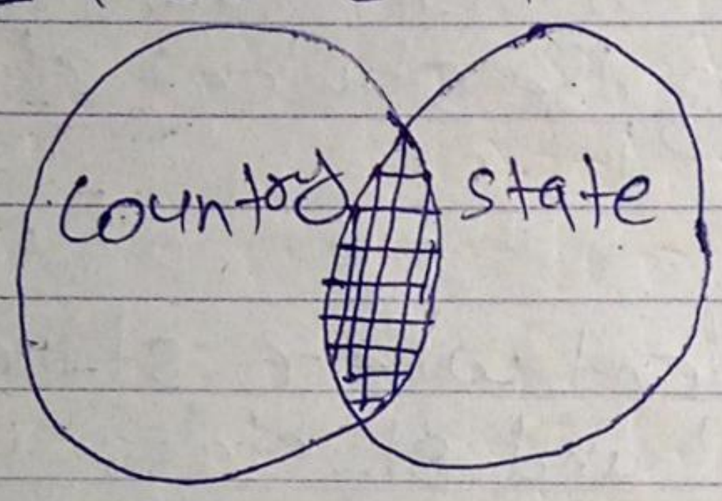
* Python MySQL Joins Complete Introduction

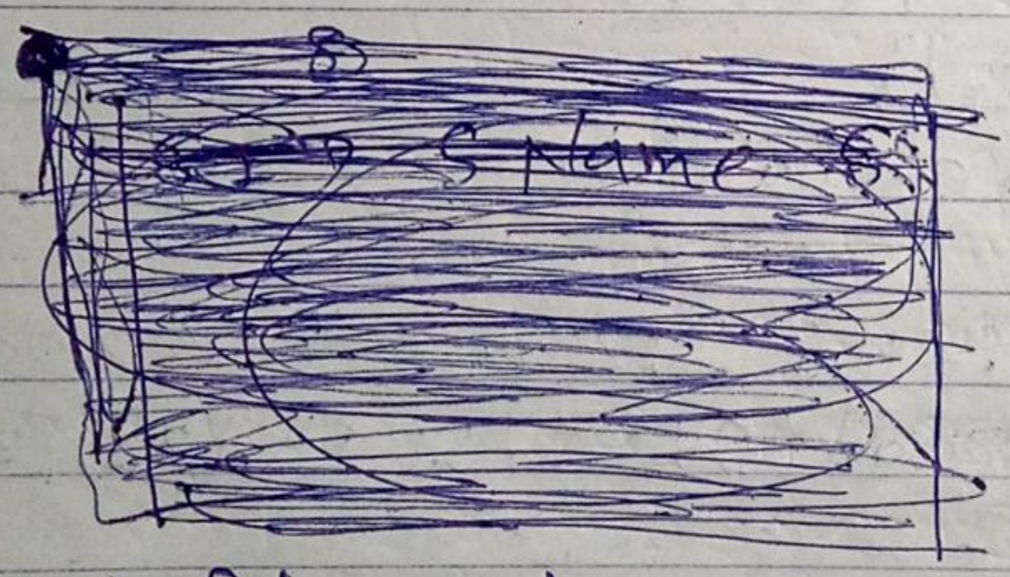
⇒ Types of Join :-

- Inner Join
- Left Join
- Right Join
- equi Join

:

- Inner Join

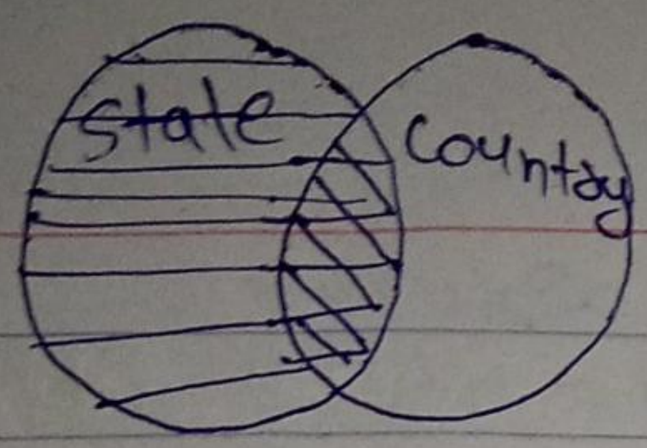




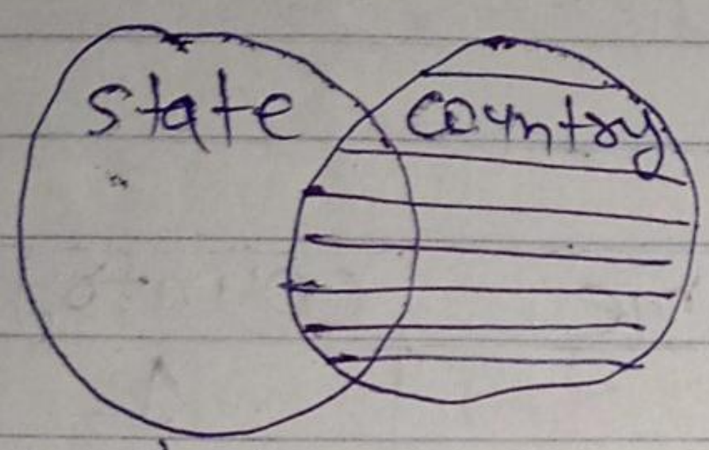
- left Join

State		
S.ID	S.Name	CID
1	Delhi	1
2	Rajhistan	1

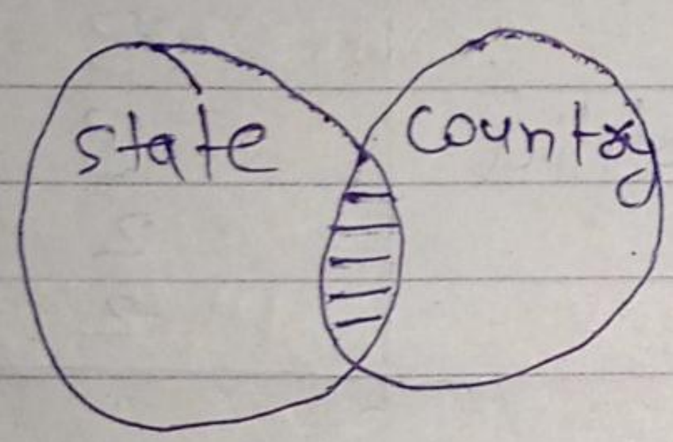
Country	
CID	C.Name
1	India
2	Srilanka



- Right Join

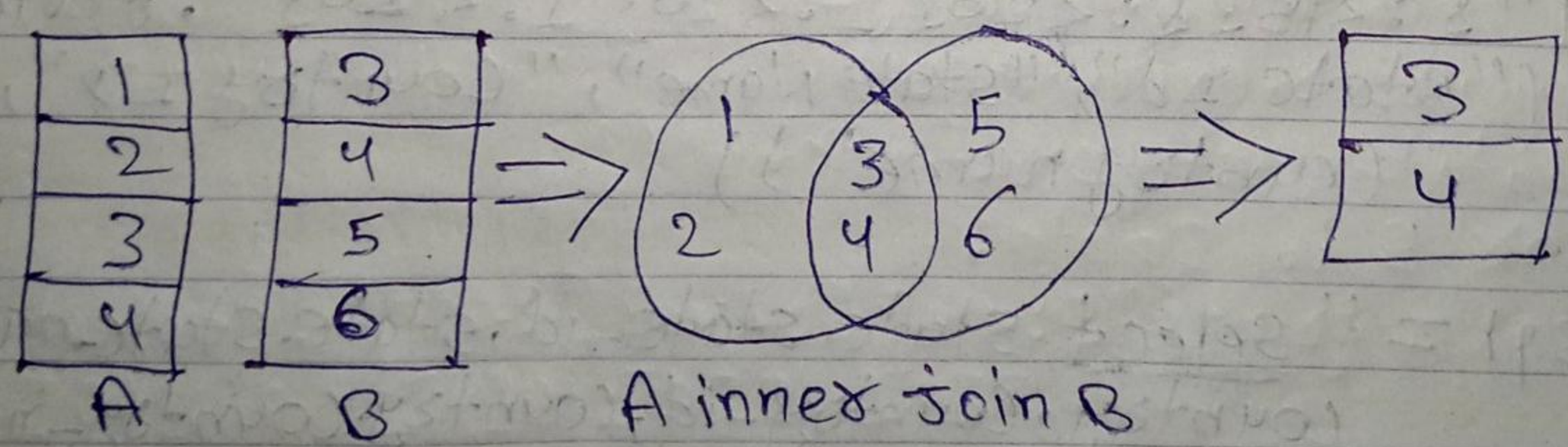


- equi Join



- MySQL Inner Join

⇒ The INNER JOIN selects all row from both participating tables as long as there is a match between the columns



Country table

country_id	country_name
1	India
2	Sri Lanka
3	Australia

state table

state_id	~~state~~ state_name	country_id
1	Rajasthan	1
2	Delhi	1
3	Victoria	3
4	Western Australia	3
5	Colombo	2
6	Kandy	2

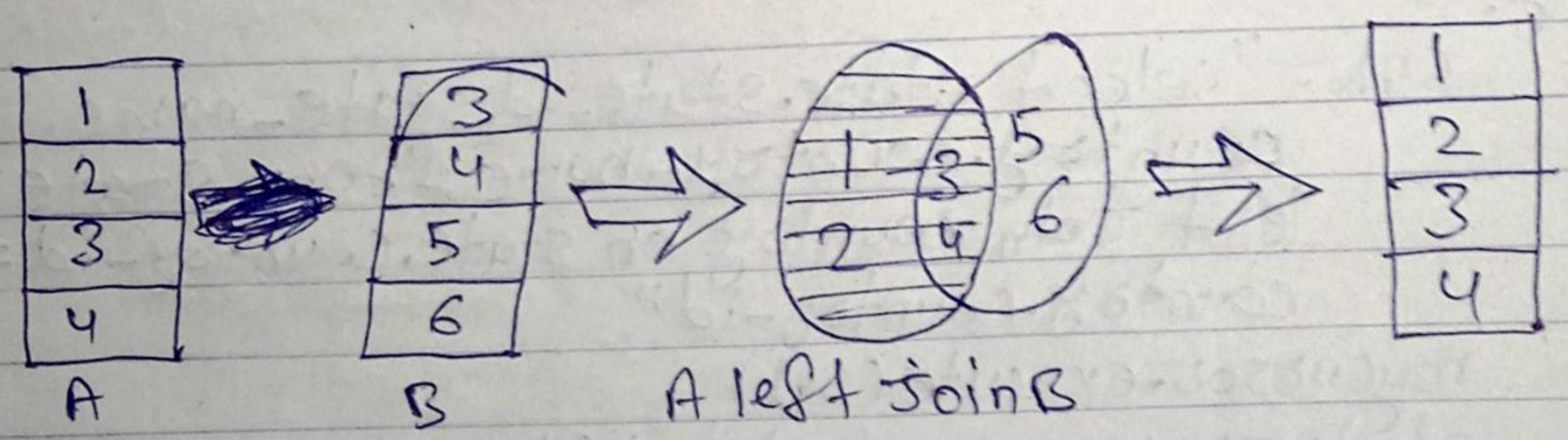
⇒ Create a filename: innerJoin.py

```
import pymysql as mq
mysql = mq.connect("localhost", "root", " ",
                   "school")
mycursor = mysql.cursor()
print("{:<15} {:<20} {:<20} {:<20}".format
      ("state Id", "state Name", "Country ID",
       "Country name"))
try:
    sql = " select state.state_id, state.state_name,
      country.country_id, country.country_name
      from state inner join country on
      state.country_id = country.country_id"
```

```
mycursor.execute(sql)
sdata = mycursor.fetchall()
for s in sdata:
    #print(s)
    print("{:<15} {:<20} {:<20}
            {:<20}".format(s[0],
            s[1], s[2], s[3]))
except:
    print("Error")
```

- MySQL Left Join

⇒ The LEFT JOIN Keyword returns all records from the left table (A), and the matched records from the right table (B).



Country Table	
Country_id	Country_name
1	India
2	Sri lanka
3	Australia

state Table

state_id	state_name	country_id
1	Rajasthan	1
2	Delhi	1
3	Victoria	3
4	Western Australia	3
5	Colombo	2
6	Kandy	2

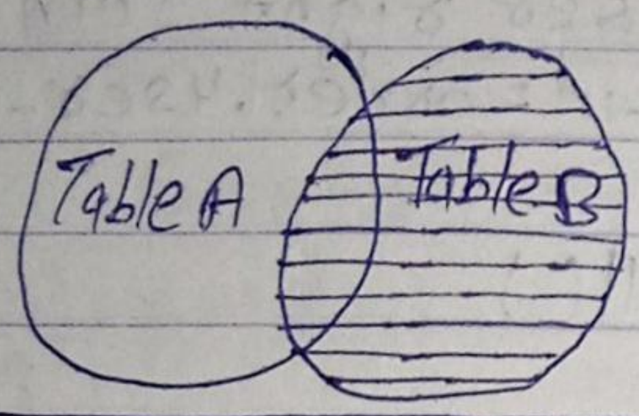
⇒ create a filename: leftjoin.py

```
import pymysql as mq
mysql = mq.connect("localhost", "root", " ", "school")
mycursor = mysql.cursor()
print("{:<15} {:<20} {:<20}".format("State Id",
      "state name", "country name"))
try:
    sql = "select state.state_id, state_name,
          country.country_name from state
          left join country on state.country_id =
          country.country_id"
    mycursor.execute(sql)
    sdata = mycursor.fatchall()
    for s in sdata:
        print("{:<15} {:<20} {:<20}".format
              (s[0], s[1], str(s[2])))
except:
    print("Error")
```

MySQL RIGHT JOIN

⇒ This means that a right join returns all the values from the right table, plus matched values from the left table.

Right Join



User Table

user_id	user_name	user_address
1	Ravi	Jodhpur
2	Ram	Jaipur

Order Table

order_id	order_date	user_id	order_amount
1	2018-12-13	1	5000
2	2018-12-13	1	2000
3	2018-12-13	2	6000
4	2018-12-13	2	4000

⇒ Create a filename:- rightjoin.py

```
import pymysql as mq
mysql = mq.connect("localhost","root","","school")
mycursor = mysql.cursor()
```

```
print("{:<15} {:<20} {:<20} {:<20} {:<20}"
    .format("user Id", "User name","user Address",
            "order Id", "order Amount"))

try:
    sql="select user.user_id,user.user_name,
        user.user_address,order.order_id,order
        .order_amount from user right join
        order on user.user_id=order.user_id"
    mycursor.execute(sql)
    sdata=mycursor.fetchall()
    for s in sdata:
        # print(s)
        print("{:<15} {:<20} {:<20} {:<20} {:<20}"
            .format(s[0],s[1],s[2],s[3],s[4]))

except:
    print("Error")
```

MySQL EQUI JOIN

⇨ Equi join returns the matching column values of the associated tables. It uses a comparison operator in the WHERE clause to refer equality.

State Table		
state_id	state_name	country_id
1	Rajasthan	1
2	Delhi	1
3	Victoria	3
4	Western Australia	3
5	Colombo	2
6	Kandy	2

Country Table	
Country-id	country-name
1	India
2	Sri Lanka
3	Australia

⇒ Create a filename:- equi-join.py

```
import pymysql as mq
mysql = mq.connect("localhost", "root", "",
        "school")
mycursor = mysql.cursor()
print("{:<15} {:<20} {:<20}".format("state Id",
        "state Name", "country Id"))
try:
    sql = "select * from state, country where
        state.country-id = country.country-id"
    mycursor.execute(sql)
    sdata = mycursor.fetchall()
    for s in sdata:
        print("{:<15} {:<20} {:<20}".format(s[0],
```

```
s[1], s[2])) s[1], s[2]))
except:
    print("Error")
```

⇒
```
import pymysql as my
mysql = my.connect("localhost", "root", " ",
                   "school")
mycursor = mysql.cursor()
print("{:<15} {:<20} {:<20}".format("state.id",
    "state name", "country Name"))
try:
    sql = "select state.state_id, state.state_name,
        country.country_name from state,
        country where state.country_id
        = country.country_id"
    mycursor.execute(sql)
    sdata = mycursor.fetchall()
    for s in sdata:
        print("{:<15} {:<20} {:<20}".format
            (s[0], s[1], s[2]))
except:
    print("Error")
```

- MySQL Self Join and Full Outer Join

⇒ (1) Self Join:-

A self join is a regular Join, but the table is joined with itself.

```
SELECT column_name(s)
FROM table1 T1, table1 T2
WHERE condition;
```

~~Catiogery~~ categories		
cat_id	cat_name	Parent_id
1	Mens	0
2	women	0
3	shirt	1
4	shoes	1
5	Jewellery	2

(2) Full Join

⇒ The SQL FULL JOIN combines the result of both left and right outer Join.

Note: MySQL doesn't support FULL JOIN

```
SELECT table1.user_name, table2.order_amount
FROM user as table1
FULL JOIN orders as table2
ON table1.user_id = table2.user_id;
```

→ Create a filename: self.Join.py

```
import pymysql as mq
db=mq.connect(host="localhost",user="root",
              "password" = "",db="ecom")
cur = db.cursor()
try:
    sql = "select * from categories as C1;
          Categories as C2 where C1.cat-id
          = C2.parent-id"
    cur.execute(sql)
    sdata = cur.fetchall()
    for a in sdata:
        print(a)
except:
    pass
```

* MySQL BETWEEN Operator

⇒ SELECT * FROM state WHERE state_id between 1 and 4

State table		
state_id	state_name	country_id
1	Rajasthan	1
2	Delhi	1
3	Victoria	3

4	Western Australia	3
5	Colombo	0
6	Kandy	2

⇒ output = Display output upto 1 to 4 of state_id

⇒ Create a filename :- between.py

```
import pymysql as mq
mysql = mq.connect("localhost","root","",
                   "school")
mycursor = mysql.cursor()
print("{:<15} {:<20}".format("state_id",
        "state Name"))
try:
    sql = "select state_id, state_name from
        state where state_id between 2 and
        5"
    mycursor.execute(sql)
    sdata = mycursor.fetchall()
    for s in sdata:
        print("{:<15} {:<20}".format
            (s[0],s[1]))
except:
    print("Error")
```

* Use of MIN and MAX Functions in SQL

=> Order Table

order_id	order_date	user_id	order_amount
1	2018-12-13	1	5000
2	2018-12-14	4	2000
3	2018-12-15	2	6000
4	2018-12-16	5	4000
5	2018-12-17	2	4000
6	2018-12-17	3	4000
7	2018-12-19	2	4000

=> Create a filename :- min-max.py

```
import pymysql as mq
mysql=mq.connect("localhost","root","","school")
mycursor=mysql.cursor()
print("{:<15}".format("order Total"))
try:
    sql="select min(order_amount) from
        order"
    mycursor.execute(sql)
    sdata=mycursor.fatchall()
    for s in sdata:
        print("{:<15}".format(s[0]))
except:
    print("Error")
```

⇒ Create a filename: maximum.py

```
import pymysql as mq
mysql=mq.connect("localhost", "root", "", "school")
mycursor=mysql.cursor()
print("{:<15}".format("order Total"))
try:
    sql="select max(order_amount) from
        order"
    mycursor.execute(sql)
    sdata=mycursor.fatchall()
    for s in sdata:
        print("{:<15}".format(s[0]))

except:
    print("error")
```

* MySQL GROUP BY statement

⇒

emp ~~department~~ Table

emp~~no~~id	e~~n~~name	d~~n~~name
11	ADAMS	CLERK
2	ALLEN	SALESMAN
6	BLAKE	MANAGER
7	CLARK	MANAGER
13	FORD	ANALYST
12	JAMES	CLERK
4	JONES	MANAGER
9	KING	PRESIDENT

5	MARTIN	SALESMAN
14	MILLER	CLERK
8	SCOTT	ANALYST
1	SMITH	CLERK
10	TURNER	SALESMAN
3	WARD	SALESMAN

⇒ Create a filename:- group-by.py

```
import pymysql as mq
mysql= mq.connect("localhost","root","","school")
mycursor =mysql.cursor()
print("{:<15} {:<15}".format("Emp Count","Emp
                                DName"))
try:
    sql ="select count(*),dname from emp
        group by dname"
    mycursor.execute(sql)
    sdata =mycursor.fetchall()
    for s in sdata:
        print("{:<15} {:<15} ".format(s[0],
                                s[1]))
except:
    print("Error")
```

* MySQL: Distinct keyword

⇒ Table is same that is on:-
⊛ MySQL Group by statement

⇒ Create a filename: distinct.py

```
import pymysql as mq
mysql = mq.connect("localhost","root"," ","school")
mycursor = mysql.cursor()
print("{:<15}".format("Emp DName")
try:
    sql = "select distinct(dname) from emp"
    mycursor.execute(sql)
    sdata = mycursor.fetchall()
    for s in sdata:
        print("{:<15}".format(s[0]))

except:
    print("Error")
```

* MYSQL SUM() Function

⇒

order Table

order_id	order_date	user-id	order_amount
1	2018-12-13	1	5000
2	2018-12-14	4	2000
3	2018-12-15	2	6000
4	2018-12-16	5	4000

5	2018-12-17	2	5000
6	2018-12-17	3	4000
7	2018-12-19	3	4000

⇒ Create a filename:- sum-mysql.py

```
import pymysql as mq
mysql=mq.connect("localhost", "root", " ", "school")
mycursor=mysql.cursor()
print("{:<15}".format("Order Amount"))
try:
    sql="select sum(order_amount) from orders"
    mycursor.execute(sql)
    sdata=mycursor.fetchall()
    for s in sdata:
        print("{:<15}".format(s[0]))
except:
    print("Error")
```

* MYSQL - AVG() Function

⇒ Table is same that is on:

(*) MysQL-SUM() Function ....

⇒ Create a filename: avg-mysql.py

```
import pymysql as mq
db=mq.connect(host="localhost",user="root",
                    password="",db="school")
cur=db.cursor()
print("{:<15}".format("Order Avg"))
try:
    sql="select avg(order_amount) from order"
    cur.execute(sql)
    sdata=cur.fetchall().
    for a in sdata:
        print(a[0])
except:
    pass
```

*MYSQL-COUNT() Function

⇒ Count rows:-

⇒ Table is same that is on:

(*) MYSQL-SUM() Function

⇒ Create a filename:- count_mysql.py

```
import pymysql as mq
db=mq.connect("localhost","root","","school")
cur=db.cursor()
print("{:<15}".format("Total Count"))
try:
    sql="select count(*) from order"
    cur.execute(sql)
```

```
        sdata = cur.fetchall()
        for a in sdata:
            print(a[0])
    except:
        pass
```

~~* MySQL - IN() or NOT() Operators~~

⇒

*MYSQL - IN() or NOT() Operators

⇒ Table is same that is on:
(*) MYSQL - Sum() Function

⇒ Create a file name: in_mysql.py

```
import pymysql as mq
mysql = mq.connect("localhost", "root", " ",
        "school")
mycursor = mysql.cursor()
print("{:<15} {:<15} {:<15}".format("order Id",
        "order Date", "order Amount"))
try:
    sql = "select order_id, order_date, order_amount
        from order where order-id in (1,6)"
    mycursor.execute(sql)
    sdata = mycursor.fetchall()
    for s in sdate:
        print("{:<15} {:<15} {:<15}".format(s[0],
            str(s[1]), s[2]))
```

```
except:
        print("Error")
```

⇒ Create a filename: notin_mysql.py

```
import pymysql as mq
mysql = mq.connect("localhost", "root", " ", "school")
mycursor = mysql.cursor()
print("{:<15} {:<15} {:<15}".format("Order Id",
        "Order Date", "Order Amount"))
try:
        sql = "select order_id, order_date, order_amount
                from order where order_id not in(1,6)"
        mycursor.execute(sql)
        Sdata = mycursor.fetchall()
        for s in sdata:
                print("{:<15} {:<15} {:<15}".format(s[0],
                        str(s[1]), s[2]))
```

COMPLETE

Assignment Based Task

* Task - 1. ⇒ Managing Driver Information

- Driver
  → did, name, licenceno

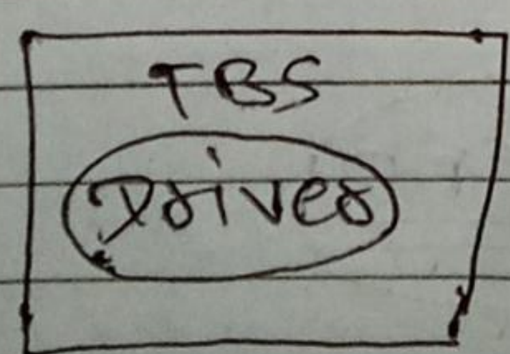
- Task
  (a) Save
  (b) Search
  (c) Edit
  (d) Delete
  (e) Display All

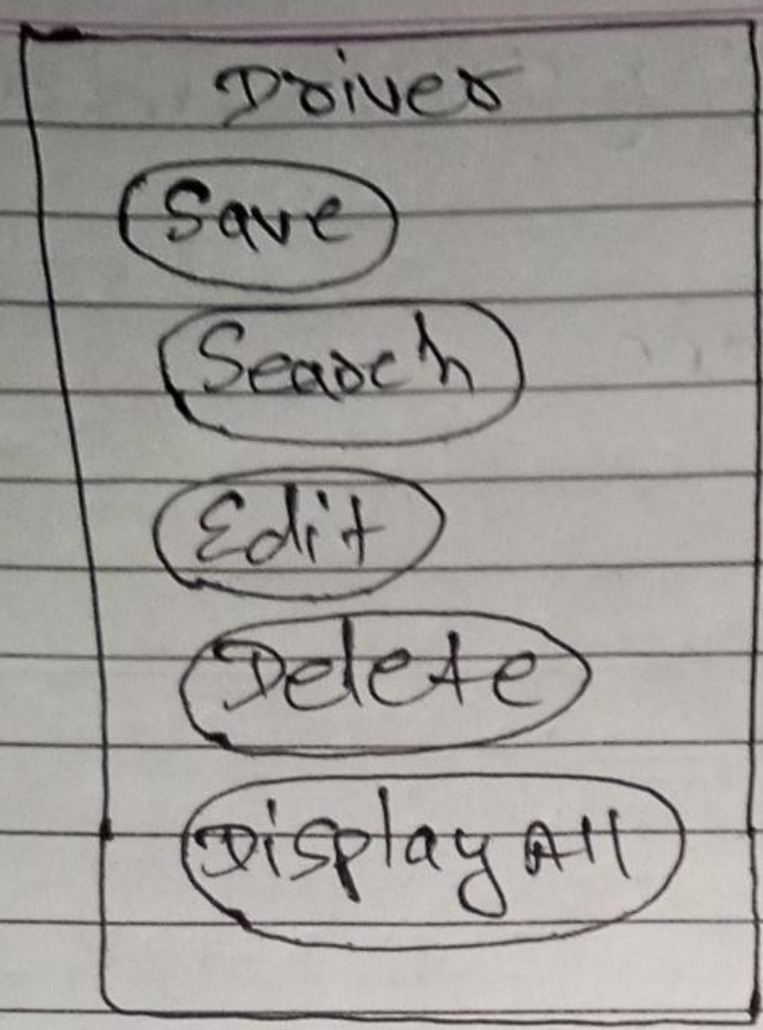
Solution,

(A) Common Tasks
  (1) Use Case ✓
  (2) Database Design ✓
  (3) Class diagram ✓
  (4) Activity diagram

(B) Individual Tasks
  (1) UI / UX
  (2) Source code
  (3) Test

(A) Common Tasks.
  (1) Use Case - Incomplete



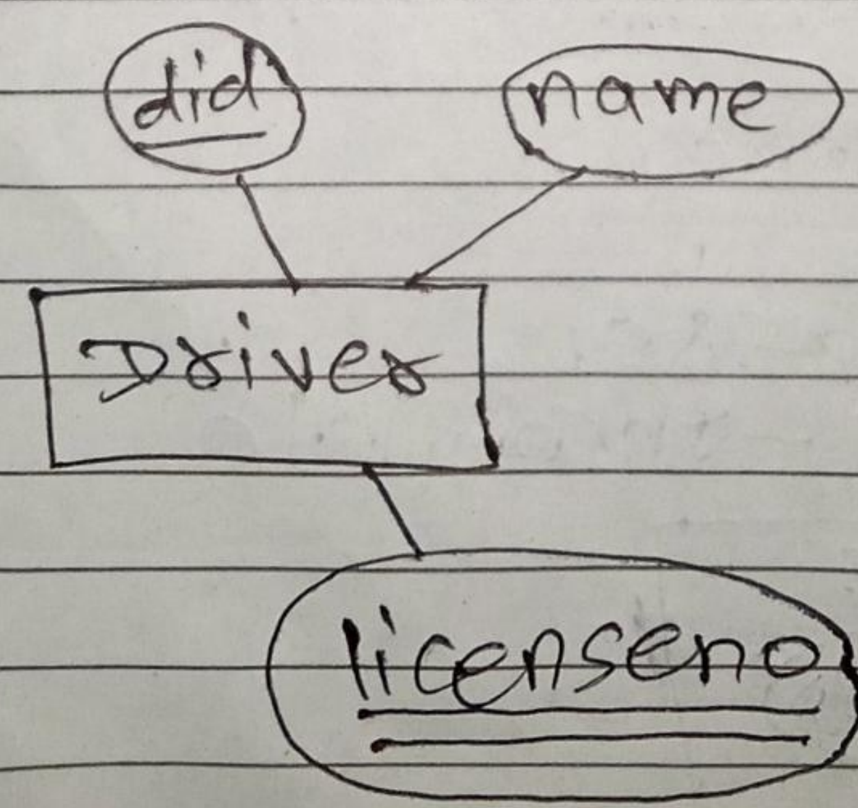


(2) Database Design

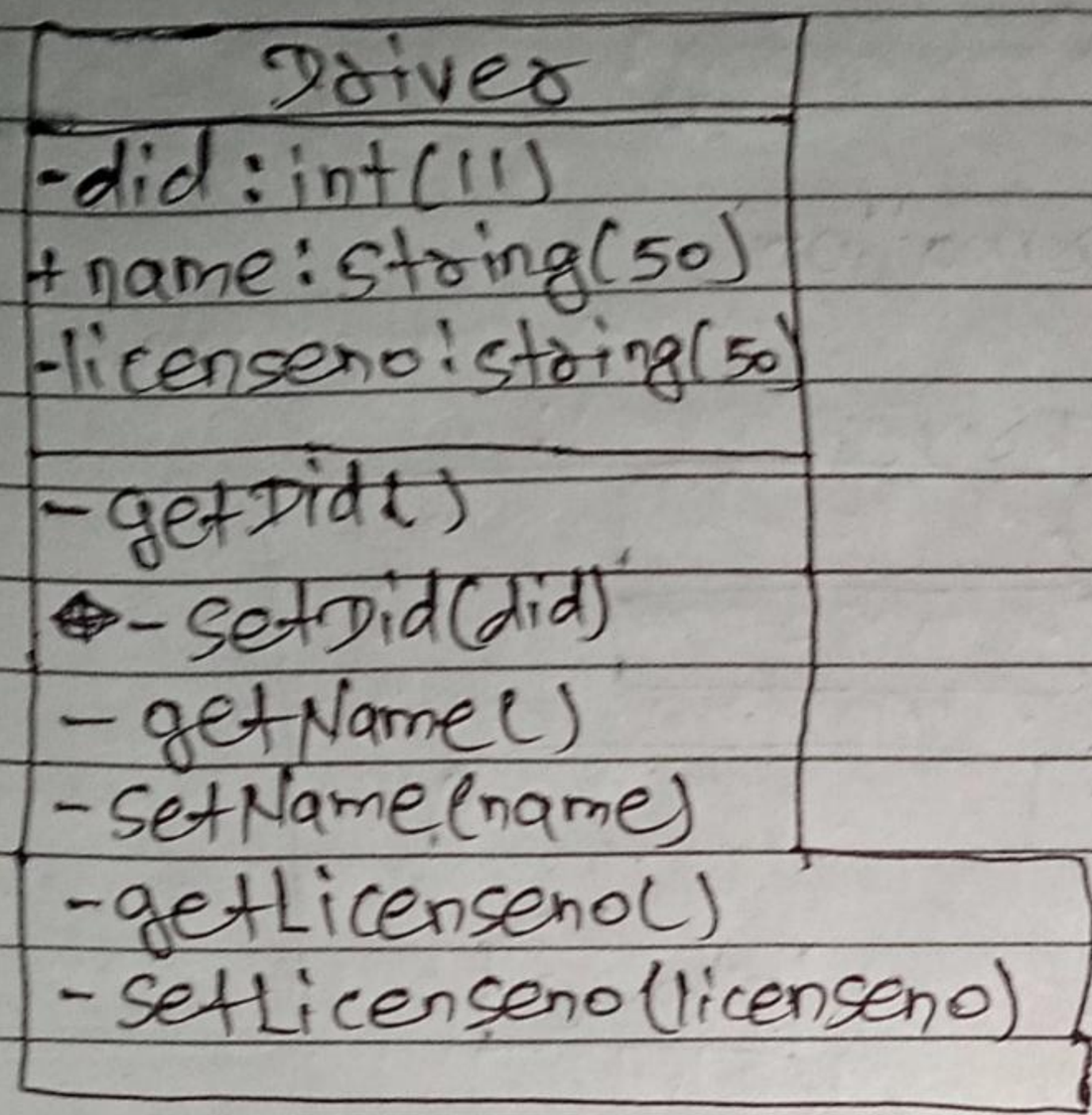
(a) Data Dictionary

SN	Field Name	Type	Length	Null	Key	Description
1	did	int	11	Not	PK	store did
2	name	varchar	50	Not	—	store name
3	licenseno	varchar	50	Not	UK	store licenseno

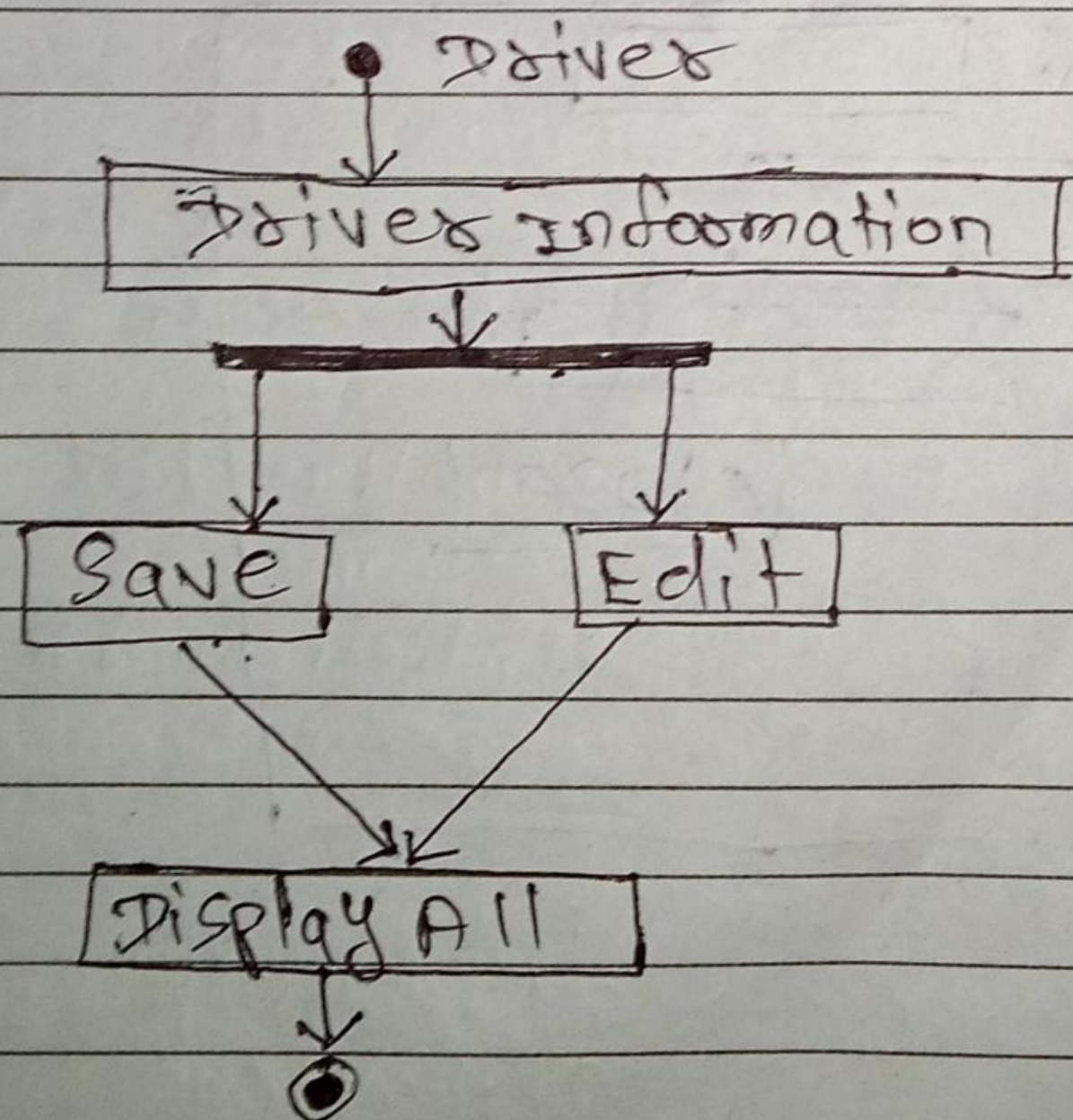
(b) ER model



(3) Class Diagram



(4) Activity Diagram



(B) Individual Tasks

(1) UI / UX

⇒

- Save information

Save Driver

Driver ID : [ ]
Name : [ ]
License No : [ ]

[Save] [close]

- Search information

Search Driver

Driver ID : [ ]
Name : [ ]
License No : [ ]

[Search] [close]

- Edit information

Edit Drivers

Driver ID: [ ]

Name: [ ]

License No: [ ]

[Save] [close]

- Delete information

Delete Driver

Driver ID: [ ]

License No: [ ]

[Delete] [close]

- Display All Information

Display All Drivers

Driver Name: [ ]

[DisplayAll] [close]

(2) Source code

⇒

Filename :- driver.py

```
class Driver():
    def __init__(self,did=0, name=None, licenseno
                =None):
        self.did = did
        self.name=name
        self.licenseno=licenseno

    #Getters
    def getDid(self):
        return self.did
    def getName(self):
        return self.name
    def getLicenseno(self):
        return self.licenseno

    #Setters
    def setDid(self, did):
        self.did=did
    def setName(self,name):
        self.name=name
    def setLicenseno(self,licenseno):
        self.licenseno=licenseno.
    #str
    def __str__(self):
        return str(self.did)+", "+self.name+
            ", "+self.licenseno
```

Date
Page

Filename:- db_Connector.py

```
import mysql.connector

class db_connector():
    connect_db = mysql.connector.connect(
                host='localhost',
                user='root',
                password='',
                database='levelyb')
```

Filename:- ManageDriver.py

```
import db_Connector
import sys
from driver import Driver

def saveDriver(driverInfo):
    Conn= ~~db_connector()~~ None
    sql = """ INSERT INTO drivers VALUES
        (%s,%s,%s)"""
    Values = (driverInfo.getDid(),
              driverInfo.getName(),
              driverInfo.getLicenseno())

    try:
        Conn = db_connector()
        cursor = Conn.cursor()
        Cursor.execute(sql, values)
        Conn.commit()
        cursor.close()
        conn.close()
```

```
        print("Save Driver!")
    Except:
        print("Error:", sys.exc_info())
    finally:
        del values
        del sql
        del conn

def searchDriver(licenseno):
    Conn=None
    sql = """ SELECT * FROM drivers
        WHERE licenseno = %s """
    values = (licenseno.getLicenseno())

    try:
        conn=db-connector()
        cursor=conn.cursor()
        cursor.execute(sql, values)
        cursor.close() => cursor.fetchall()
        conn.close()
        print("Driver Searched")
    except:
        print("Error:", sys.exc_info())
    finally:
        del sql
        del conn
```

```
def updateDriver(driverInfo):
    conn = None
    sql = """ UPDATE drivers SET
        name = %S, licenseno = %S WHERE
        did = %S"""
    values = (driverInfo.getDid(),
              driverInfo.getName(),
              driverInfo.getLicenseNo())
    try:
        conn = db_connector()
        cursor = conn.cursor()
        cursor.execute(sql, values)
        conn.commit()
        cursor.close()
        conn.close()
        print("data updated")
    except:
        print("Error:", sys.exc_info())
    finally:
        del values
        del sql
        del conn

def deleteDriver(did):
    conn = None
    sql = """ DELETE FROM drivers WHERE
        did = %S"""
    values = (did.getDid())
```

```
    try:
        conn = db_connector()
        cursor = conn.cursor()
        cursor.execute(sql, values)
        conn.commit()
        cursor.close()
        conn.close()
        print("delete driver successfully")
    except:
        print("Error:", sys.exc_info())
    finally:
        del values
        del sql
        del conn

def allDrivers():
    conn = None
    sql = """SELECT * FROM drivers"""

    try:
        conn = db_connector()
        cursor = conn.cursor()
        cursor.execute(sql)
        cursor.fetchall()
        conn.
        cursor.close()
        conn.close()
        print("data show all")
```

```
except:
        print("Error:", sys.exc_info())
finally:
        del sql
        del conn
```

File name: Test.py

```
from driver import Driver
from ManageDriver import *
```

Test-1

```
did = name = lno = None
did = int(input("Enter ID:"))
name = input("Name:")
lno = input("License No:")
d1 = Driver(did, name, lno)
saveDriver(d1)
```

Test-2

```
~~did~~ lno = None
lno = input("License No:")
d2 = Driver(lno)
searchDriver(d2)
```

Date
Page

Test-3

```
did=name=lno=None
did=int(input("Enter ID:"))
name=input("NAME:")
lno=input("License No:")
d3=Driver(did,name,lno)
updateDriver(d3)
```

Test-4

```
did=None
did=int(input("Enter ID:"))
d4=Driver(did)
updateDriver(d4)
```

Test-5