CHAPTER 1

"REVISION OF THE BASIC OF PYTHON"

Camlin Page
Date / /

- Python is a powerful and high-level language.
- It is an interpreted language.
- It is widely used for general purpose higher level programming developed by Guido van Rossum in 1991.

- It has two basic modes :-
  - Interactive mode (immediately returns results)
  - Script mode (create file with (.py) extension and save it).

BASIC PYTHON PROGRAM :-

(1) Print ("HELLO WORLD")
⇒ O/P :- HELLO WORLD

(2) a = 10
b = 20
c = a+b
Print (c)
d, e = 2, 3
Print (d, e)
⇒ O/P :- 30
2 3

(3) name = "Aisha"
age = 22
Salary = 30,000
Print ("The age of Aisha is 22 and Salary is 30000

[P.T.O]

(04) name = (Shyam)
Print = (name)
⇒ O/P :- Shyam.

(05) name = input ()
Print (name)
⇒ O/P :- Enter name Suchismita
Suchismita.

(06) name = input ("Enter your name":)
print = ("my name is :", name)
⇒ O/P :- Enter name : Mohak
my name is : Mohak

KEYWORDS :-

- Keywords are the reserved words used in python programming for some specific purpose.

E.g :- and, del, not, while, elif, global, or, with, if, else, break, continue, def, return, for, try.

- True, False and None are the keywords must written as above except these 3 all other keys are written in lowercase.

03/04/2021

OPERATORS IN PYTHON :-

- Python language supports a rich set of operators and the type of operators are :-

→

01:- ARITHMETIC OPERATORS:-

Assume a = 7 and b = 3 (operands).

	OPERATORS	DESCRIPTION	EXAMPLE
1.	'+'	Addition	a+b = 10.
2.	'-'	Substraction	a-b = -4.
3.	'*'	multiplication	a*b = 21.
4.	'/'	Division	a/b = 2.3
5.	'**'	Exponent	a**b = 343.
6.	'%'	modulus	a%b = 1.
7.	'//'	floor division	a//b = 2.

02:- RELATIONAL OPERATORS:-

Assume a = 4 and b = 7.

	OPERATORS	DESCRIPTION	EXAMPLE
1.	'<'	Less than	a<b = True
2.	'>'	greater than	a>b = false
3.	'<='	less than equal to	a<=b = false
4.	'>='	greater than equal to	a>=b = false
5.	'!='	Not equal to	a!=b = True
6.	'=='	Equality check	a==b = false

03:- LOGICAL OPERATORS:-

	OPERATORS	DESCRIPTION
1.	'AND'	a and b
2.	'OR'	a or b
3.	'NOT'	a not b

[P.T.O]

04:- ASSIGNMENT OPERATORS :-

Assume a = 5.

	OPERATORS	DESCRIPTION	EXAMPLE
1-	'='	Assigning.	a = 5
2-	'+='	add then Assign.	a = a+5
3-	'-='	Subtract then Assign.	a = a-5
4-	'*='	multiply then Assign.	a = a*5
5-	'/='	Divide then Assign.	a = a/b
6-	'%='	Take module then Assign.	a = a%5
7-	'**='	Take exponent then Assign.	a = a**b
8-	'//='	Take floor division then Assign.	a = a//5

05:- BITWISE OPERATORS :-

a = 0011 1100 b = 0000 1101.

	OPERATORS	DESCRIPTION	EXAMPLE
1-	a&b (binary AND)	a & b	a&b = 0000 1100
2-	a|b (binary OR)	a | b	a|b = 0011 1101
3-	a^b (Bitwise XOR)	a ^ b	a^b = 0011 0001
4-	~a (NOT)	~a	~a = 1100 0011

06:- MEMBERSHIP OPERATORS :-

OPERATORS	DESCRIPTION	EXAMPLE
in	if value in sequence	Returns true
not in	if not in a sequence	Returns true

→

Date / /

07:- IDENTITY OPERATORS :-

	OPERATORS	DESCRIPTION	EXAMPLE
1	is	if id(x) is equal to id(y)	Returns true.
2	is not	if id(x) is not equal to id(y)	Returns true.

CW 05/04/2021

CONTROL STATEMENTS IN PYTHON :-

- Three types of programming :-
  - Sequential
  - conditional
  - Repeatational

CONDITIONAL :-

01) if
02) if --- else
03) multiple if else
04) Nested if else

REPEATATIONAL :-

1) for i in range (starting value, ending value, step):
while loop :- i = 0 (initialization)
while <condition>:
incrementation

Q- write a python program to input a number and check whether the number is positive, Negative or zero.

⇒ num = int(input("Enter a no:"))

[P.T.O]

```
if (num == 0):
    print (num, "is zero")
elif (num < 0):
    print (num, "is negative")
else:
    print (num, "is positive").
```

Q- for odd and Even :-

```
num = int (input ("Enter a no.":))
if (num % 2 == 0):
    print (num, "is even")
else:
    print (num, "is odd")
```

- In python we may have one more type of statement i.e, jump statement.

- BREAK STATEMENT.
- CONTINUE STATEMENT.
- PASS STATEMENT.

Program :-

```
for i in range (1, 11)
if (i == 3):
    print ("hello", end = ' ')
    continue.
if (i == 8)
    break
if (i == 5)
    pass
```

→

LIST IN PYTHON:-

- It is a collection of items and each item has its own index value. The index value of first item is zero (0); from leftwards and (-1) from rightward.
- The len() of the list is no. of items.

```
                (-5) (-4) (-3) (-2) (-1)
Q- List = [10, 20, 30, 40, 50]
           (0)  (1)  (2)  (3)  (4)
```

- Positive index = (4).
- negative index = (-5).
- length = (5).

CREATING A LIST:-

```
          (0)  (1)  (2)    (3)  (4)   (5)  (6)
a = [10, 20, 'abc', 30, 3.14, 40, 50]
print(a)
for i in range(0, len(a)):
    Print(a[i], end = ' ')
print('\n')
for i in range(-len(a)-1, -1, -1):
    print(a[i], end = " ")
print('\n')
for i in a[::-1]:
    Print(i, end = " ")
print('\n')
for i in reversed(a):
    print(i, end = " ").
```

O/P:- a = [10, 20, 'abc', 30, 3.14, 40, 50]

- 10, 20 'abc' 30 3.14 40 50
- 50 40 3.14 30 'abc' 20 10
- 50 40 3.14 30 'abc' 20 10

[P.T.O]

• 50 40 3.14 30 'abc' 20 10

AW 06/04/2024

TUPLE IN PYTHON:-

- It is a sequence of immutable objects.
- It's just like a list which contains a sequence of data items with its index value.
- Difference between list and tuple is that Tuple can't be changed like a list.
- List uses square bracket i.e, [] and Tuple uses round bracket i.e, ().

• [] ⟶ Mutable ⟶ List = [1,2,3,4,5]
• () ⟶ Immutable ⟶ Tuple = (1,2,3,4,5)

CREATING A TUPLE AND ACCESSING ITS ELEMENTS:-

```
a = (10,20,'abc',30,3.14,40,50)
print(a)
for i in range(0,len(a)):
    print(a[i],end=" ")
print('\n')
for i in range(-len(a),-1,-1):
    print(a[i],end=" ")
print('\n')
for i in range a[::-1]
    print(i,end=" ")
print('\n')
for i in reversed(a):
    print(i,end=" ")
```

O/P:- (10, 20, 'abc' 30 3.14 40 50)
10 20 'abc' 30 3.14 40 50
50 40 3.14 30 'abc' 20 10 ⟶

50 40 3.14 30 'abc' 20 10

FUNCTIONS IN TUPLE:-

- Tuple(seq):- This is used to convert a list into tuple.
- min(tuple):- Returns minimum value from tuple.
- max(tuple):- Returns maximum value from tuple.
- len(tuple):- Returns total length of tuple.
- cmp(tuple 1, tuple 2):- Compares elements in a Tuple.

PREDICT O/P FOR THE FOLLOWING:-

```
- for i in range (1,10,3):
      print (i)
```

O/P:- 1
4
7

07/04/2021

Q- x = hello world

```
Print (x[:2], x[:-2], x[-2:])
```

O/P:- he
hello wor
ld

Q2:- Print (x[6], x[2:4])
O/P:- w ll

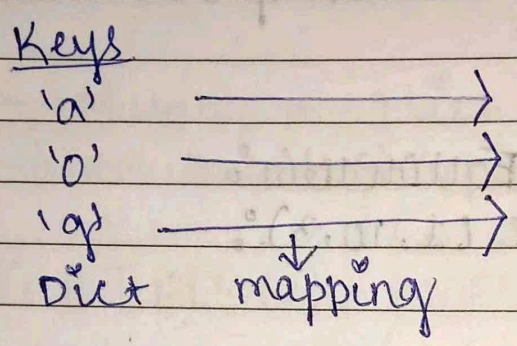
Q3:- Print (x[2:-3], x[-4:-2])
O/P:- llo wo or

[P.T.O]

DICTIONARY IN PYTHON:-

-Dictionary is an unordered collection of data values in form of a pair of keys and values. having a (:) in between them. and all the pairs are separated by a (,) comma and kept within flower bracket.

eg:- dict = {"a": "alpha", "o": "omega", "g": "gamma"}



#create an empty dictionary

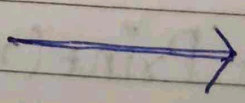
```
Dict = {}
Print (Dict)
```

O/P:- {}

#Adding elements

```
Dict[1] = 'Aisha'
Dict[2] = 'Govind'
Dict[3] = 'Arya'
Print [Dict].
```

O/P:- Dict = {1: 'Aisha', 2: 'Govind', 3: 'Arya'}



08/04/2021

write the o/p for following code.

⇒ marks = {}
Marks [Sanat] = 75
Marks [Anita] = 85
Marks [Binaya] = 90
print (marks)

O/p = Marks = {'Sanat': 75, 'Anita': 85, 'Binaya: 90}

2:- update the marks dictionary with the following.
Marks ['prachi'] = 80

O/p = Marks = {'Sanat': 75, 'Anita': 85, 'Binaya': 90, 'prachi: 80'}

ACCESSING ELEMENTS IN A DICTIONARY:-

D = {1: 'AAA', 2: 'BBB', 3: 'CCC'}

```
for i in D:
    print (i, end = " ")
for i in D:
    print (D[i], end = " ")
for K,V in D.items():
    print (K, V, end = " ")
for i in D.keys():
    print (i, end = " ")
for i in D.values():
    print (i, end = " ").
```

K = Keys
V = Values.

O/p = 123
AAA BBB CCC
1AAA 2BBB 3CCC
123

[P.T.O]

AAA BBB CCC

APPLICATION BASED PROGRAM:-

01- Write the output for the following code:

(a)
```
values = []
for i in range (1,4):
    values.append (i)
print (values
```

O/P:- [1,2,3]

(b)
```
for x in range (35, 10, -5):
    print (x)
```

O/P:- 35
30
25
20
15

02- Rewrite the code after correcting it:

(a)
```
if N=>0
    print(odd)
else
    print("even")
```

Sol^n:-

O/P:-
```
N= input(int("Enter a no:"))
if N>=0:
    print("odd")
else:
    Print("even")
```

(b) for = 20
for 1 = 50 :
for 3 = for + for 1 + for 2
print (for 3)

Sol^n :-
O/P :- X = 20
Y = 50 (:) invalid
for 3 = x + y
print (for 3).

- (:) id error
- for can't be used as it's a keyword.

Q3- Find the invalid identifier or variable and write the reason of it.

1 :- abc @ shyam
2 :- for = 10
3 :- __Raju
4 :- 123 arya
5 :- a = 5

Sol^n :- (3) 1) abc @ shyam = Invalid.
- As, a variable name can only contain alpha-numeric characters and underscores.
- @ is a special symbol.

2) for = 10 = invalid.
- As we can't use any keyword as an identifier and for is a keyword.

[P.T.O]

3) __Raju = Invalid
- As, a variable name can only contain alpha_numeric characters and underscores but at beginning we can't use.

4) 123aryan = Invalid
- Digits can be used but not in the beginning.

5) a = 5 = valid.

(W) 12/04/2021

IDENTIFIERS RULE:-

- No keywords can be used as identifier.
- Can use 79 characters space.
- Special symbols are not allowed.
- __ could be used but not in beginning.
- digits upto (0-9) can be used but not in beginning.
- a-z & A-Z can be used.

Q1- What are None - Literal in python?
- None is a legal empty value used in python which indicates something has not yet created.

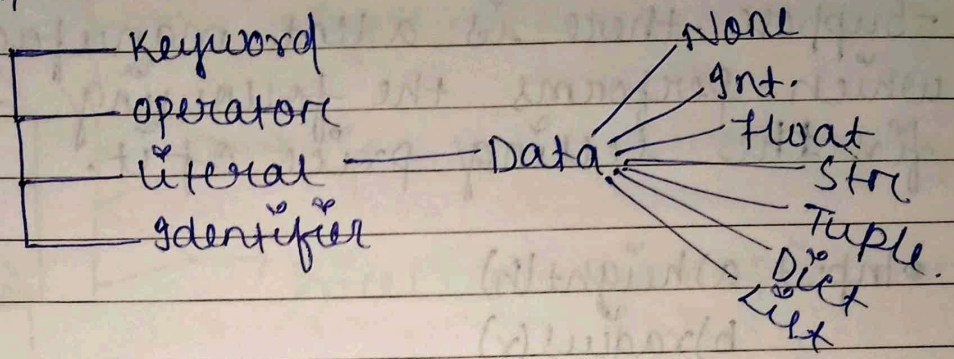
Q2- Can list be used as a keys of a dictionary?
- NO, list can't be used as they are mutable & keys of a dictionary are immutable.

→

Camlin Page
Date 12 / 04 / 2021

BASIC BUILDING BLOCKS OF PYTHON :-

01) Comments
02) Variables
03) Constant
04) Tokens (Smallest unit.



—— x ——

[P.T.O]

CHAPTER:- 2

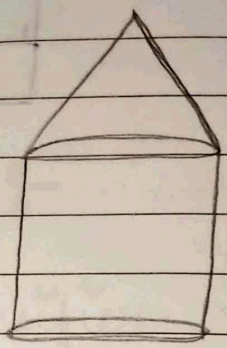
"FUNCTIONS"

Camlin Page
Date 12/04/2021

-When the program size increases, its difficult to manage.

*Consider the following examples:

-Suppose, there is a tent manufacturing company which performs the following task to fix the selling price of it.



- Input a) height(h)
  b) radius(r)
  c) Slant height(l)
  for the tent.

-Calculate area of tent
-Calculate the cost for making tent.
-Calculate the net payable amount by 18% tax.

#Program to calculate the area and net payable amount for tent

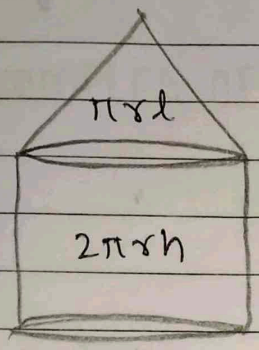
```
h=float(input("Enter height:"))
r=float(input("Enter radius:"))
l=float(input("Enter slant height:"))
for the tent.
con_area=3.14*r*l
cyl_area=2*3.14*r*h
total_area=con_area+cyl_area
unit_price=float(input("Enter cost of cloth per meter:"))
total_cost=unit_price*total_area
tax=0.18*total_cost
```

→

Date 12/04/2021

net_price = total_cost + tax
Print("Amount payable by user":, net_price)

15/04/2021 Second approach to solve the same problem known as modular approach / programming



→ con(r,l) calculate area of conical part.
→ cyl(r,h) calculate area of cylindrical part.
→ Price() calculate netprice with tax of 18%

- Modular programming is the process of dividing a computer program into separate independent blocks of code with different names and specific functionalities.

ADVANTAGES OF FUNCTION:-

- Increases ~~read~~ readability and easy to understand as program is better organized
- Reduced code makes:
  1) Debugging easier (Error checking & correct)
- Increases reusability by avoiding writing the same code repeatedly.
- parallely work could done by distributing among team members.

[P.T.O]

- To achieve modular programming one of the way is to use function which avoids writing repeated code by simply calling the func in a program and importing the func to any other programs, once func is defined.

```
- def cyl(r,h):
    area_cyl = 2*3.14*r*h
    return(area_cyl)

- def con(r,l):
    area_con = 3.14*r*l
    return(area_con)

- def price(cost):
    tax = cost*0.18
    net_price = cost+tax
    return(net_price)

h = float(input("Enter height:"))
r = float(input("Enter radius:"))
csa_cyl = cyl(h,r)

l = float(input("Enter slant height:"))
csa_con = con(r,l)

Total_area = csa_cyl+csa_con
print("Total area of tent:","Total_area)
unit_price = float(input("Enter cost of cloth per meter:"))
total_cost = unit price * Total_area
print("Net payable amount",price(total cost:))
```

→

021 CW 19/04/2021

Camlin Page
Date 19/04/2021

TYPES OF FUNCTION:-

- Inbuilt func^n
- module func^n
- user defined func^n

- EXAMPLES OF IN-BUILT FUNC^N:-
  - print()
  - input()
  - len()

- EXAMPLES OF MODULE:-
  - squareroot()
  - area()

- USER DEFINED FUNC^N:-
  - we create func^n as per our requirement.
- creating a func^n consists up two things:-
  - func^n definition
  - func^n calling

SYNTAX OF FUNCTION:-

- def <func^n name>(parameter):

eg:- # to add two numbers using func^n with no parameter and return value

```
def add():
    a=int(input("Enter a:"))
    b=int(input("Enter b:"))
    c=a+b
    print(c)
add()
```

[P.T.O]

O/P :-

```
Enter a : 5
Enter b : 6
11.
```

#Program to find sum of first n number without func^{n} :-

```
n = int(input("Enter numbers : "))
Sum = 0
for i in range (1, n+1):
    Sum = Sum + 1.
print (Sum)
```

O/P :-

```
Enter number : 5
1-5
```

#Program to find sum of first n numbers using func^{n} :-

```
def Sumnum(n):
 Sum = 0
 for i in range (1, n+1):
     Sum = Sum + 1
  print (Sum)
num = int (input ("Enter n :"))
Sumnum (num)
```

FUNCTION CREATE :-

- function def (Parameter)
- function calling (Argument)
- id(parameter) = id(Argument)

Date 19/04/2022

#program to find area of a rectangle using func^n with parameter :-

```
def rec(l,b):
    area l*b
    print(area)
len = float(input("length:"))
bre = float(input("breadth:"))
rec(len,bre)
```

W 22/04/2021

DEFAULT PARAMETER :-

- python allows to assign default value to the parameter which values are decided and couldn't be changed during func^n call.

- The default parameters are already set at the time of func^n definition and doesn't have argument during func^n call

#Program which accepts your name as an input and display by adding hello before your name as a default parameter :-

```
def name(n,str='hello'): #defining func
    msg = str + " " +n
    print(msg)
fullname = input("Enter your name as n value:")
str = "hello"
name(fullname,str) # func^n call.
```

[P.T.O]

O/P :-
Enter your name as n value : Riya
hello Riya.

- In the written program str is a default parameter whose value is fixed as hello. which can't be changed during function call.

- A default parameter can be overwritten during func" call.

program which accepts your name as an input and display by adding hello before your name as a default parameter which can be overwritten.

```
defname(n, str = "hello")
    msg = str + " " n.
        print(msg)
fullname = input("Enter name as n value)
str = input("Enter string:")
name(fullname, str)
```

O/P :-
Enter your name as n value :- Mohan
hello mohan.

23/04/2021

program to find factorial of a number using user defined func" with parameter

```
def factorial(num):
    fact = 1
    for i in range(1, num+1): →
```

Date 23/04/2021

```
        fact = fact * i
        print(fact)
n = int(input("Enter no.:"))
factorial(n)
```

O/P :-

Enter no. : 5
1
2
6
24
120

#write a program to find factors of a number using user defined func^n # with parameter :-

```
def factor(num):
    for i in range(1, num+1):
        if num % i == 0:
            print(i, "factor of no.", num))
n = int(input("Enter no:"))
factor(n)
```

O/P :- Enter no : 6
1 factor of no. 6
2 factor of no. 6
3 factor of no. 6
6 factor of no. 6

#write a program to check whether a no. is prim or composite using user defined func^n with parameter

[P.T.O]

```
def prime(num):
    count=0
    for i in range(1,num+1):
        if num%i==0:
            count+=1
    if count==2:
        print(num,"is prime").
    else:
        print(num,"is composite').
number=int(input("Enter no:")
prime(number)
```

O/P:-

```
Enter no:5
5 is prime
```

#write a program to find area of square by using user defined func^n with parameter

```
def sq(side):
    area=side*side
    print(area)
s=float(input("Enter side:"))
sq(s).
```

O/P:-

```
Enter side:4
16
```

→

04/2021

CW 24/04/2021

Camlin Page
Date 24/04/2021

POSITIONAL PARAMETER (Required Argument)

- when the arguments are passed to a parameter are in correct position or order are known then they can be called as positional parameters.

```
#program to display name and age

def fun(name, age): # parameters passed in order
    print("Name":name)
    print ("Age": age)
fun name = "mohak", age=15) # argument passed in same order.
```

O/P :-

```
Name : Mohak
Age : 15
```

KEYWORD ARGUMENTS :-

- During func" call we may change the order of parameters which is passed as an argument.
- The arguments are called and identified by the parameter name rather than its order.

```
def fun(name, age):
    print("Name:", name)
    Print("Age:" age)
fun(age=15, name="mohak") # parameters as an argument are not in same order.
```

[P.T.O]

- fun def (parameters) a, b = formal
  - numerical
  - string
  - List
  - Tuple
  - dictionary

- All these could be passed as parameters in func^n def.
  - They are data types / objects.
- func^n call (arguments) = actual parameter

MUTABLE/IMMUTABLE PROPERTIES OF DATA OBJECT:-

- Everything in python is an object and every object is either mutable or immutable.

- when an object is initiated that it is assigned as a unique object Id and once, set can be or can't be changed.

MUTABLE:-

- It can be changed after creation.

eg:- List
dict
set
byte array.

IMMUTABLE:-

- It can't be changed after creation.

eg:- int, float, complex, string, Tuple.

→

Camlin Page
Date 26 / 04 / 2021

PASSING STRING TO A FUNCTION:-

-we could also pass string as a parameter to a func^n and string is immutable type. To demonstrate this above a program is given below:

```
def funcStr(Str):
    Str = "hi"
    print("Str:", Str)
    print(Id(Str))
Str = "hello"
funcStr(Str)
print("Str after update:", Str)
print(Id(Str))
```

O/P :-

```
Str:hi
31346592.
Str after update:hello
36799072.
```

-In the above program we could see that once the string assigned with a value or it is created. we couldn't change it if we assign it with a different value a new different object is created, which could proved by Id().

Str = 'hi' = 31346592
Str = 'hello' = 36799072

[P.T.O]

Date 26/04/2021

PASSING TUPLE AS A PARAMETER TO A FUNCTION:-

- We could also pass tuple as a parameter to a func^n and Tuple is Immutable type. To demonstrate this above a program is given below:

```
def funtpl(tuple):
    tuple t=(1,2)
    print(tuple)
    print(Id(tuple))
t=(3,)
funtple(t)
print(t)
print(Id(tuple))
```

O/P:-

```
(3,1,2)
36510056
(3,)
36798992
```

- After updating/changing the tuple they won't remain the same.
- Hence, we can't change a tuple once it's created.

→

Date 26/04/2021

PASSING LIST AS A PARAMETER TO A FUNCTION:-

- List also can be passed as a parameter to a func^n and list are mutable and to demonstrate this above a program is given below:

```
def funlist(list):
    list[10,20]
    print(list)
    print(id(list))
funlist(list)
list=[1,2,3,4,5]
print(list)
print(id(list))
```

O/P :-

```
[10,20]
34994336
[1,2,3,4,5]
34994336
```

- List are mutable as after updat^n they have same id.
  - list[10,20] = 34994336
  - list[1,2,3,4,5] = 34994336

PASSING DICTIONARY AS A PARAMETER TO A FUNCTION:-

- The dictionary can be passed as a parameter to a func^n like other variable are passed.

[P.T.O]

28/04

- Dictionary is mutable. To demonstrate this a program is given below:

```
def fundict(d):
    d["c"]=3
    print(d)
    print(Id(d))
mydict={"a":1,"b":2}
print(mydict)
```

O/P:-

```
{'a':1,"b":2}
  36457376
{'a':1,'b':2,'c':3}
  36457376
```

CW 27/04/2021

RETURN VALUE:-

- Syntax:-

```
def funname(parameter):
    ----------
    return result
func^n call
```

- Here, the parameter and returning value is optional.

- without parameter. without return value.
- with parameter without return value.
- without parameter with return value.
- with parameter with return value.

→

CW 28/04/2021

Date 28/04/2021

#program to demonstrate user defined func^n with parameter and with return value :-

```
def sum(x,y):
    z = x+y
    return z
a = int(input("x:"))
b = int(input("y:"))
r = sum(a,b)
print(r)
```

O/P :-

```
x:4
y:5
9
```

FUNCTIONS RETURNING VALUE :-

- A func^n may or may'nt return a value when called
- The return statement returns the values from the func^n.

- The return statement does the following :-

  - Returns the control to the calling func^n.
  - Return Value() or None

USING RETURN :-

- print result - return result
- func^n call = Variable and print the value

[P.T.O]

29/04/20

#Program to calculate area and perimeter of a rectangle by using user defined func" with parameter & with return value :-

```
defrec(l,b):
    area=l*b
    return area
    perimeter=2(l+b)
    return perimeter
len=float(input("length:"))
bre=float(input("breadth:"))
r=rec(l,b)
print(r).
```

#program to calculate area of a rectangle by using user defined func" with parameter and with return value :-

```
defarea(l,b):
    ar=l*b
    return(ar)
len=float(input("length:"))
bre=float(input("breadth:"))
a=area(len,bre)
print(a)
```

O/P :-

```
length:4
breadth:5
20.0
```

→

FUNCTION RETURNS NONE VALUE:-

- Sometimes a func^{n} performs calculat^{n} and not return any value or returns None value are called void func^{n}.

#Func^{n} with no parameter perform some operat^{n} and return None is a void func^{n}.

```
def sum():
    return
r=sum()
print()
```

O/P :- None.

CW 30/04/2021

FLOW OF EXECUTION:-

- flow of execut^{n} can be defined as the order in which the statement in a program are executed. The python interpreter starts executing the instruct^{n} in a program from the first statement.

- The statements are executed one by one, in the order of appearance from top to bottom.

- when the interpreter encounters a func^{n} definition, the statements inside the func^{n} are not executed until and unless the func^{n} is called.

[P.T.O]

#program to understand the flow of execut^n using func^n.

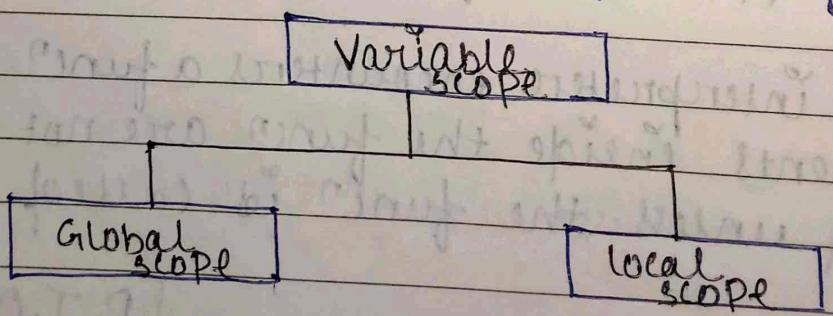
```
-print using func^n's
 hello python()          # func^n call.

def hellopython():       #func^n definition
    print("I Love programming")
```

- on executing the above code the following error is produced which we know is the execut^n starts from func^n calling but still if we write the program in above sequence we found error because the func^n definition always should preside the func^n calling that's the reason why we found error.

SCOPE OF A VARIABLE :-

- Every variable has a well-defined accessibility. The part of the program where a variable is accessible can be defined as the scope of that variable.
- A variable can have the following scopes :-



→

- A variable that has global scope is known as a global variable.
- A variable that has a local scope is known as local variable.

GLOBAL VARIABLE:-

- A variable that is defined outside any func^n or any block is known as a global variable

- It is accessible anywhere in the program.

- Any changes made to the global variable will put impact to the whole program.

LOCAL VARIABLE:-

- A variable that is defined inside any func^n or any block is known as a local variable.

- It is accessible only in the func^n or block where it is defined.

- Any changes made to the local variable only affects to the func^n or block where it is defined.

#global and local variables:-

```
num=5
def funcnum():
    Y=num+5
    print(num,"is global variable")
    print(Y,"is local variable")
funcnum()
```

[P.T.O]

Print(num, "is a global variable")
print(y, "is a local variable")

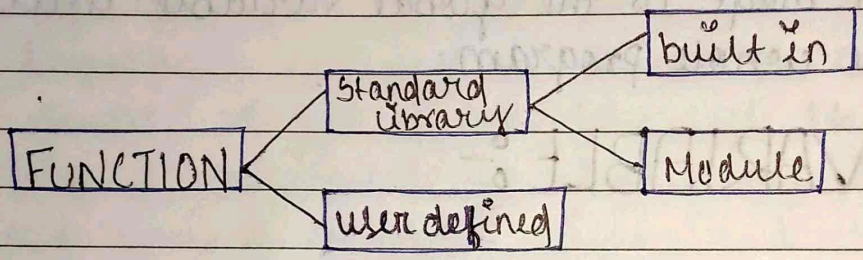
O/P:-

5 is a global variable
10 is a local variable
5 is a global variable

>>> Print(y, "is local variable")
NameError: name 'y' is not defined

01/03/2021

PYTHON STANDARD LIBRARY:-



BUILT-IN FUNCTION:-

-Already we know that user defined func^n which gets familiar with built in func^n. To know this better examples are given below:-

Input or output	func^s	mathematical func^s	other func^s
input()	~~type~~	abs()	__import__()
print()	~~print~~	divmod()	len()
	str()	max()	range()
	int()	min()	type.
	float()	pow()	
	tuple()	sum()	
	~~range~~	oct()	→

Date 01 / 05 / 2021.

COMMONLY USED BUILT-IN FUNCTION WITH EXAMPLE :-

- abs(x) :- It returns absolute value of x.

for eg :-

```
>>> abs(-5.7)
5.7
```

- divmod(x,y) :- It returns a tuple of (quotient, reminder).

for eg :-

```
>>> divmod(-7,2)
(-4,1)
```

- Pow(x,y[,z]) :- It returns $(x^y)\%z$.

for eg :-

```
>>> Pow(5,2,4)
1
```

MODULE :-

- A number of functions grouped together to form a module.
- A complex problem mayn't be feasible to manage the code in one single file so the program is divided into different parts under different levels called modules
- A module is created as a python file (.py). containing a collec^n of func^n definition which can be saved under a module and reused.

[P.T.O]

- To use a module we need to import the module. The syntax of import statement is :-

- import Modulename1 [, modulename2, ...]

- To call a func^n of a module, the func^n name should be proceeded with the name of the module with a (.) dot as a separator.

Syntax :-

modulename.funcname()

BUILT IN MODULE :-

- Python library has many built-in modules which are really easy to ~~used~~ be used.

SOME COMMONLY USED MODULES :-

Math module :- It contains different type of mathematical func^{n}s.

03/05/2021

SOME COMMONLY USED MATH MODULE :-

1) math.ceil(x) :- It returns the ceiling value of x

eg :-

```
>>>math.ceil(9)
9
```

2) math.floor(x) :- It returns the floor value of x

eg :-

```
>>>math.floor(4)
4.
```

→

Date 03/05/2021

3) math.abs(x) :- It returns absolute value of x.

eg :-

```
>>> math.abs(-4)
4.0
```

4) math.factorial(x) :- It returns factorial of x.

eg :-

```
>>> math.factorial(5)
120
```

5) Math.gcd(x, y) :- It returns gcd of x & y.

eg :-

```
>>> Math.gcd(10,2)
2
```

6) Math.sqrt(x) :- It returns squareroot of x

eg :-

```
Math.sqrt(.64)
0.8
```

CW 04/05/2021

STRING FUNCTIONS :-

01) capitalize() :- converts the first character to upper case.

02) casefold() :- Converts str. to lowercase.

03) centered() :- Returns a centered string.

04) count() :- Returns the no. of time a specified value occurs in a str.

[P.T.O]

05) encoded() :- the sequence of code points converted into a set of bytes.

CREATING USER DEFINED MODULE :-

*BASIC MATHS :

- This contains the following user defined func^n.

a) create a user defined module Basic math that contains the following user defined functions

b) To add two numbers and to return their sum.

c) To substract two numbers and to return their difference.

d) To multiply two numbers are return their products.

e) To devide two numbers and return their quotient. and print "division by 0" error if the denominator is 0.

f) Also add a docstring to describe the module, import and execute func^n's.

CW) 05/05/2021

```
""" """ """
#Beginning of module
defaddnum(x,y):
    return(x+y)
defsubnum(x,y):
    return(x-y)
defmultnum(x,y):
```

→

```
    return(x*y)
def divnum(x,y):
    if y==0:
        print("Division by zero Error")
    else:
        return(x/y)
                                    #End of module.
```

O/P :-

```
#Statements for using module basic_math
>>>import basic_math
#Display descriptions of the said module
>>>print(basic_math.__doc__)

    basic_math module
>>>a=basic_math.addnum(2,5)
>>>a
  7
>>>a=basic_math.subnum(2,5)
>>>a
 -3
>>>a=basic_Math.multnum(2,5)
>>>a
 10
>>>a=basic_math.divnum(2,5)
>>>a
 0.4
>>>a=basic_math.divnum(2,0)
Zero divide Error
```

——— Y ———

[P.T.O]

CHAPTER 7

"USING PYTHON LIBRARIES"

Date 07/05/2021

- Python Programs are the collecⁿ of methods, classes which can be used easily

- Python program is made using :-
  - Library or package.
  - Module.
  - funcⁿs / sub-modules

RELATION BETWEEN PYTHON-LIBRARIES, MODULE AND PACKAGE :-

MODULE :-

- A module is a file containing python definition, funcⁿs, variables, classes and statements.

- The extension of this file is ".py".

PACKAGE :-

- Collecⁿ of python modules in a directory (folder) is known as python package.

LIBRARY :-

- Collecⁿ of many packages in python are called python library.

→

Camlin Page
Date 07/05/2021

MODULE IN PYTHON AND ITS ADVANTAGES:-

- In last chapter we have discussed about different in-built module for eg :- math module as well as user defined module.

ADVANTAGES:-

- All the func^n's of a module can be used anywhere in any program after importing the module.
- Reusability.
- It helps in logically organizat^n of python code.
- Easy programming.

EXAMPLE OF USER DEFINED MODULE:-

create a user defined module with area_calculate module name and create func^n's according to below specifiat^n's :-

- create func^n to calculate area of a square.
- create func^n to calculate area of a rectangle.
- create func^n to calculate area of a circle.
- create func^n to calculate area of a cylinder.

O/P :-

```
""
area_calculate
""
def area_sq(side):
    return side * side
def area_rect(l,b):
    return l * b
```

[P.T.O]

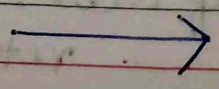
```
def area_circle(r)
    return 3.14*r*r
def area_cylinder(r,h)
    return 2*3.14*r*h

-import area_calculatn
a=area_calculatn.area_sq(5)
a
25
b=area_calculatn.area_rect(5,3)
b
15
c=area_calculatn.area_circle(5)
c
78.5
d=area_calculatn.area_cyl(5,7)
d
219.8
```

IMPORT MODULE BY USING FROM-STATEMENT:-

-If we are using from statement to import a module and want to call a funcn under the module, we need to follow the below procedure with given syntax:-

*Import the module by using syntax;
    from module_name import *
- call funcn under this module by using syntax,
from module_name import funcn_name.



Date 07/05/2021

for eg :-

for the written program, we have to use :-

```
from area_calculat import area_sq
print (area_sq(4))
16.
```

10/05/2021 PACKAGE / LIBRARY :-

- Group of instruc^{n} ⟶ func^{n}
- Group of func^{n} ⟶ module
- Group of modules ⟶ Library

- Python package is the collec^{n} of same type of module.
- we may have in-built packages
- package / Library are generally same.

12/05/2021 CREATING IN-BUILT PACKAGES :-

STEPS TO CREATE A PACKAGE :

01) To create a package with name geometry by creating a folder or directory with the same name geometry.

02) The package geometry contains two module files named as :-

- area_geometry
- perimeter_geometry

03) In the same directory we will put another file named as "__init__.py"

[P.T.O]

04) __init__.py is necessary because this is the file which tells python that directory is package. This file initializes the package.

05) Then we import the package and use the content.

EXAMPLE :-

- Let's create a package named as geometry. In which area_geometry and perimeter_geometry will be created.

- In module area_geometry there are group of func^n's to calculate area of square, rectangle, circle.

- In module perimeter_geometry there are group of func^n's to calculate perimeter of square, rectangle, circle.

STEP 1 :- To create geometry folder directory. go to the location or path.
for eg :-
(C:\Users\Lenovo\AppData\Local\Programs\Python\Python37-32\ directory path).

STEP 2 :- create module 1 area_geometry.
eg :-

```
""
area_geometry
""
def area_sq(x):
    return x*x
def area_rect(l,b):
```

→

```
    return l*b
def area_circle(r):
    return 3.14*r*r
```

- Create module2 perimeter_geometry

eg:-
```
""
def per_sq(x):
    return 4*x
def per_rect(l,b):
    return 2*(l+b)
def per_circle(r):
    return 2*3.14*r
```

- Save these modules/files by naming area_geometry.py and perimeter_geometry.py in geometry folder/directory

STEP3:- create a blank file named as __init__.py and save it in the same directory/folder.

- This __init__.py file can be left blank or it could have codes to import the modules from the package

- from geometry import area_geometry
- from geometry import perimeter_geometry

CW 13/05/2021 STEP4:- Importing the package geometry could be done in any of these following methods

METHOD 1: first import the package

Syntax:- import package_name

a=package_name.module_name.func()_____
name(parameter)

[P.T.O]

Camlin Page
Date 13/05/2021

```
>>> import geometry
>>> a=geometry.perimeter_geometry.per_rectangle(5,8)
26
```

METHOD2 :- If we want to access a specific module of a package then we can use from and import keywords.

Syntax :-

```
from package_name import module.name
a=module_name.func^n_name(parameter)
```

```
from geometry import area_geometry
>>> x=area_geometry.area_square(5)
>>> x
>>>25.
```

METHOD3 :- If a user wants to import all the modules of shape package then he/she may use *(asterisk).

Syntax :-

```
from package_name import *
a=module_name.func^n_name(parameter)
>>> from geometry import *
5=perimeter_geometry.per_square(4)
>>>5
16
```

14/05/2021

HOW TO USE A CREATED PACKAGE IN A PROGRAM :-

-steps needed to follow to use a created package in a program :

→

Date 14 / 05 / 2021

STEP 1: create a file named as main.py in the same directory where geometry package is located

STEP 2: write the following code in main.py file.

```
import geometry
from geometry import area_geometry
from geometry import perimeter_geometry
print["The area of square is", geometry.area_geometry.
area_square(4)]
print["The perimeter of rectangle is", perimeter_geometry.
perimeter_square(6,3)]
```

O/P :-

The area of square is 16
The perimeter of rectangle is 15

- create a package named shape in which rectangle, square, circle modules are placed along with their func'n to calculate area and perimeter for each objects import the created package, use the package in program. write down the steps required.

STEP 1: create a folder directory named as shape in the path :-

C:/users/Lenovo/AppData/Local/Programs/Python/Python37-32/shape-module

STEP 2: create 1. rectangle.py module.

```
def area_rect(l,b):
    return(l*b)
def per_rect(l,b):
    return(2*l+b)
```

[P.T.O]

module 2:- Square.py

```
def area_sq(a):
    return (a*a)
def perimeter_sq(a)
    return(4*a*a)
```

module 3:- circle.py

```
def area_circle(r):
    return(3.14*r*r)
def per_circle(r).
    return(2*3.14*r)
```

Wed 15/05/2024

STEP 3:- Create __init__.py file and keep it in shape directory folder

we may have this __init__.py as a blank file which initializes the package or we may have the following code in this file.

Code:-

```
from shape import rectangle
from shape import square
from shape import circle
```

- Package shape is created so, now we need to import it.

STEP 4:- Importing the package shape by:-

METHOD 1:- Import shape

```
from shape import square
a=square.per_sq(4)
a
16
```

Date 15/05/2021

METHOD 2 : Import shape
a = shape.rectangle.area_rect(6,4)
a
24

METHOD 3 : Import shape
a = circle.per_circle(3)
a
18.84

STEP 5 : using the created package in program create a main.py file and locate in the same locatn where shape package is located.

-write the following code in main.py

```
Import shape
from shape import rectangle
from shape import square
from shape import circle
print("The area of rectangle is",rectangle.area_rec(6,3))
print("The perimeter of square is", square.per_sq(3))
```

O/P :-

The area of rectangle is 18
The perimeter of square is 36

——— x ———

[P.T.O]

CHAPTER 4

FILE HANDLING IN PYTHON

-usually organisations wants to permanently store information about employees, inventory, Sales, etc. to avoid repetitive tasks of entering the same data.

-Hence, data are stored permanently on secondary storage devices for reusability.

-usually, organisations want to avoid repetitive tasks.

-We store python programs written in script mode with a .py extension.

-Each program is stored on the ~~data entered,~~ and secondary device as a file, likewise the data entered, and the output can be stored permanently into a file.

WHAT IS A FILE?-

-A file is named locat^n on a secondary storage media where data are permanently stored for later access.

TYPES OF FILES:-

-computers store every file as a collec^n of 0s and 1s i.e, in binary form

- There are two types of files:-
  a) Textfile
  b) binary file

→

Camlin Page
Date 17/05/2021

TEXT FILE :-

- It consists of human readable characters, which can be opened by any text editor.

BINARY FILE :-

- They are made up of non-human readable characters and symbols which requires specific programs to access its contents.

NEED FOR A DATA FILE :-

- To store data in organised manner.
- To store data permanently.
- To access data transfer.
- To search data faster.
- To easily modify data later on.

FILE HANDLING :-

- File handling in python enables us to create, update, read and delete the files stored on the file system through our python program
- The following operat^ns can be performed.

- In python file handling consists of :-
  a) Open the file.
  b) process the file i.e, perform read or write operat^ns.
  c) close the file

[P.T.O]

16/05/2021

TEXT FILE :-

- A text file is a sequence of characters consisting of alphabets, numbers and other special symbols.

- Text files are with extension like :-
  a) .txt.
  b) .py.
  c) .csv etc.

- when we write several lines in a text file by using any text editor, internally it is stored in form of 0s or 1s.

- The value of each character of the text file stored as bytes in ASCII UNICODE or any other encoding scheme. rather than binary file.

- So, while opening the text file, the text editor translates each ASCII value and shows the equivalent character that is readable by human being

- for eg :- The ASCII value 69 (binary equivalent 1000001) will be displayed by a text editor as the letter 'A' since the number 65 in ASCII character represents 'A'.

- Each line of a text file is terminated by a special character called the end of line (EOL).

→

Date 18/05/2021

for eg :- In python \n - for newline

- whitespace, comma (,) and tab (\t) - separates values in a text file.

- where we create a textfile by using notepad is stored in .txt file is in bytes.

- whereas .docx in KBs.

19/05/2021

EXAMPLES OF (EOL) :-

- hello\n everyone → hello
  everyone
- hello\t everyone → hello everyone
- hello\n everyone → hello school
  everyone
  everyone cool.
- hello, everyone → hello everyone

BINARY FILES :-

- Binary files are stored in terms of bytes (0s & 1s) Rather than representing the ASCII values of characters, they represent the actual content such as image, audio, video, compressed versions of other files, executable files, etc. These files are not in human readable formats.

- To open a binary file using text editor will show some garbage values.

[P.T.O]

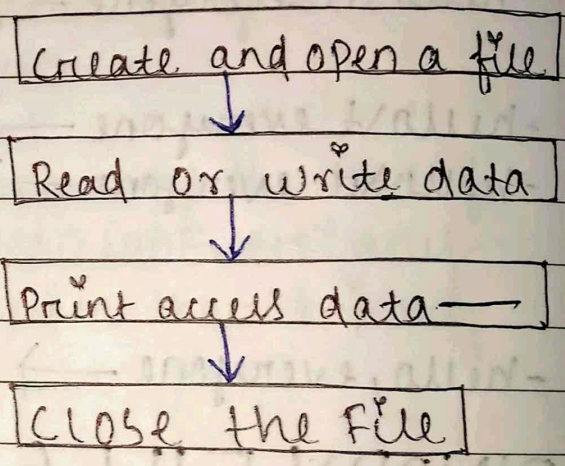
- we need specific software to read or write the contents of a binary file.

- As, the content of binary file are not in human readable, even a single bit change can corrupt the file because it is difficult to remove any error.

- we can read and write ~~any error~~ both text and binary files through python programs

WORKING WITH TEXT FILES :-

- Basic operat^n with files :-
a) Read the data from file
b) write the data to a file
c) Append the data to a file
d) Delete a file



20/05/2021

OPENING A FILE :-

- To open a file in python, we use the open() func^n. The syntax of open() is as follows :-
file_object = open(file_name, access_mode)

FILE_OBJECT :-

- It establishes a link bet^n the program and the data file stored in the permanent storage.

→

- open() func^{n} takes two parameters :-
  a) file name.
  b) mode.

- In which file name is compulsory and mode is optional

- open() returns a file object which is stored in the variable file_object

- This variable is used to transfer data to and from the file.

- file_object has certain Attributes that tells us basic informat^{n} regarding the file.

- file-object contains Attributes such as :-

  - ⟨file.closed⟩ returns true if the file is closed and false otherwise

  - ⟨file.mode⟩ returns the access mode in which the file was opened

  - ⟨file.name⟩ returns the name of the file.

FILE_NAME :

- It is the name of the file in which we perform file operat^{n}.
- If the file isn't in the current working directory, then need to specify the complete path of the file along with its name.

[P.T.O]

Date 20/05/2021

ACCESS MODE :-

- The access_mode is an optional argument which represents the processing mode of the file.

22/05/2021

FILE OPEN MODE :-

	FILEMODE	DESCRIPTION	POSITION
1.	'r'	read the file	Beginning
2.	'rb'	open file in binary mode	Beginning
3.	'r+/+r'	read & write the file	Beginning
4.	'w'	write mode which overwrites the content of file	Beginning
5.	'wb+/+wb'	read & write the binary file	Beginning
6.	'a'	Append mode add content to the file	At the end of file.
7.	'a+/+a'	read and append the file.	At the end of file.

- If file doesn't exist it creates a file and write into it.

EXAMPLE TO OPEN A FILE AND TO READ ITS CONTENT :-

file name : demofile 1.txt.

code :

```
f=open("demofile 1.txt","r")
```

→

Date 22 / 05 / 2021

eg :-
```
f = open ("demofile 1.txt", "r").
f.read()
'helloworldddd'
```

READ THE DATA FROM A FILE :-

- There are three types of func^n's to read data from a file :-

01 - read() :- reads the entire file.
02 - readline() :- reads n bytes / characters
for eg :-
```
f = open ("demofile 1.txt", "r")
f.readline(5)
'hello'
```
- where demofile 1 content is helloworld
- But readlines reads only 5 characters from the beginning as n+5.
03 - readlines() :- Reads all lines and returns a list
for eg :-
```
f.readlines()
["hello world"]
```

24/05/2021

STEPS TO READ DATA FROM A FILE :-

- create and open a file using open() func^n.
- use the read(), readline() or readlines() func^n.
- print / access the data.
- close the file.

[P.T.O]

Date 24 01 2021

EXAMPLES :-

01. Program using read func^n :-

- Consider a file named as "demofile 1.txt" and the content of the file is ⟶ Hello world!

```
fn = open("demofile 1.txt", 'r')
str = fn.read()
print(str)
fn.close()
```

O/P :-

Hello World!

02. Program using readline() func^n :-

```
fn=open("demofile 1.txt",'r')
fn.readline(5)
fn.close()
```

O/P :-

Hello

03. Program using readlines() func^n :-

```
fn=open("demofile 1.txt","r")
fn.readlines()
fn.close()
```

O/P :-

["Hello world"]

⟶

2021.

Mon 25/05/2021

Camlin Page
Date 25/05/2021.

DIFFERENCE BETWEEN TEXT FILE AND BINARY FILE:-

TEXT FILE:-

- Its Bits represents character.
- It is in human-understandable format.
- Stores only plain text in a file.
- Can be opened in any text editor.
- Mostly .txt, .rtf are used as extension to text files.

BINARY FILES:-

- Its Bits represent a custom data.
- It is not in human-understandable format.
- Can store different types of data
- Can be opened in specific applicatn.
- Can have any applicatn defined extension.

WRITE DATA TO A FILE:-

- There are two types of funcn to write the data to a file:-
- write() :- write the data to a file.

Syntax :- for text file :-

```
write(string)
```

(b) for binary file

```
write(byte string)
```

- writelines() :- write all the strings in a list as lines to file.

[P.T.O]

- To write the data to an existing file you have to use the following mode :-

"w" = write :- will overwrite any existing content

- program to overwrite a file using write() and 'w' mode.

```
f=open("demofile 1.txt",'r")
>>>str = f.read().
>>>print (str)
and welcome u all to the class
>>>f.close()
```

- Here, the demofile 1 was having above content which will be overwritten.

```
>>>f=open("demofile 1.txt","w")
>>>f.write ("Goodmorning everyone")
20
>>>f.close()
>>>f=open ("demofile 1.txt",'r')
>>>f.read()
'Goodmorning everyone'
```

27/05/2021

EXAMPLE OF Writelines() method :-

```
f=open("demofile 1.txt",'r+')
>>>f.read()
'Students of class XII welcomes U'
>>>f.writelines(['SecA, SecB"])
>>>f.close()
>>>f=open('demofile 1.txt,"r")
>>>f.read()
'Students of class XII welcomes U SecA.SecB
```

→

Camlin Page
Date 27/05/2021

APPEND THE DATA TO A FILE:-

-This operat" is used to add the data in the end of the file. It doesn't overwrite the existing data in a file. To write the data in the end of the file. you have to use the following mode:-

- "a" = append: will append to the end of the file.

EXAMPLE:-

```
f=open('demofile 1.txt','r')
>>> f.read()
'Students of class XII'
>>> f.close()
>>> f=open("demofile 1.txt","a")
>>> f.write("Welcomes u")
10
>>> f.close()
>>> f=open("demofile 1.txt","r+")
>>> f.read()
'Students of class XII welcomes u'
```

FILE OPERATION:-

01) creating file:-

- Text editor --→ open notepad --→ write the content --→ Save --→ filename . fileextention --→ .txt
- Either save it python from where you could access directly
  - Ex:- f=open("demofile 1.txt")

[P.T.O]

Date 27/05/2021

CW 29/05/2021

- otherwise the file could be saved in any other locatn and to use it we have to give the path along the file name.

Ex. fn = open("D:\Suchinotes\Assignment\demofile1.txt")

02) Opening file :-

Syntax :- file_obj = open(filename, accessmode)

Compulsory Opt

03) Reading file :-

Syntax :- file_obj = open(filename, accessmode)

file_obj.read funcn ---> read(), readline(), readlines()

or variable = file_obj.read funcn's

print(variable)

04) writing file :-

Syntax :- file_obj = open(filename, accessmode)

file_obj.write methods ---> write(). writeline()

or str = ("hello")

f.write(str).

CW 31/05

05) closing file :-

- once we are done with the read/write operatns on a file, it is a good practise to close the file.

- while closing a file, the system frees the memory allocated to it.

The syntax of close() is:

file_object.close()

Ex: f.close()

→

2. cw) 29/05/2021

Camlin Page
Date 29/05/2021

SETTING OFFSETS IN A FILE:-

-Till now we have used to access data in a random fashion, but we could access the data in a random fashion, through seek() and tell() func^n's in python.

- tell()
- seek()

THE TELL() METHOD:-

-This func^n returns an integer that specifies the current posit^n of the file object in the file.

-The syntax of tell() is :-

file_obj.tell()

cw) 31/05/2021

for eg:-

```
fout = open("story.txt","w")
fout.write("welcome python")
print(fout.tell())
fout.close()
```

O/P:-

14

eg:-

```
f = open("demofile 1.txt",'r+')
f.write("hellopython")
12
f.tell()
12.
```

[P.T.O]

Date 31/03/2022

THE SEEK() METHOD:-

- It is used to place the cursor at a particular position in a file.
- The syntax of seek() is:- (compulsory) (OPT)
  - file_object.seek(offset[,reference_point])

OFFSET:-

- offset is the place of a file where we want to put our cursor.

REFERENCE_POINT:-

- It indicates the starting posit^{n} of the file object which may have:-

0 - beginning of the file.
1 - current posit^{n} of the file
2 - end of the file

- By default the value of reference_point is 0.

for eg:-

file_obj.seek(5,0)
or file_obj.seek(5)

- Both the statements are same.

- Here the offset value is compulsory and the reference_point is optional.

→

eg:-
```
f = open("demofile 1.txt","r+")
f.write("hello python")
f.seek(5)
5
f.read()
' Python'
```

Tue 01/06/2021

RELATIVE AND ABSOLUTE PATH:-

- we all know that the files are kept in directory which are also known as folders.
- Every running program has a current directory.
- In general python programs always see the default directory first i.e, python folder itself.
- As python programs / files are in the same python folder we could use directly the files by giving ~~Absolute~~ Relative path of the file.

for eg:-
```
f=open("demofile 1.txt","r+")
f.read()
'hello python'
```

- when the files are in some other drives / folder we need to give the full path for using the file i.e, known as ~~relative path~~. Absolute path.

eg:-
```
f=open("D:\Surbhinotes\Assignment\demofile 1.txt","r")
f.read()
'Welcome u All'
```

[P.T.O]

relative path = just a file name.
absolute path = full path.

06 WORKING WITH FILES AND DIRECTORIES:-

- os module provides many funcs which can be used to work with files and directories.

- getcwd() is a very useful func which can be used to identify the current working directory (cwd).

```
>>> import os.
>>> os.getcwd()
C:\\Users\\Lenovo\\AppData\\Local\\Programs\\Python\\Python37-32
```

STANDARD FILE STREAMS:-

- we use standard I/O streams to get better performance from different I/O devices.
- Some standard streams in python are as follows:-
  - Standard input stream sys.stdin.
  - Standard output stream sys.stdout.
  - Standard error stream sys.stderr.

03/06/2021

STANDARD INPUT, OUTPUT AND ERROR STREAMS IN PYTHON:-

- Most programs make output "standard out", input from "standard in", and error messages go to standard error. Standard output is to monitor and standard input is from - keyboard.

→

Date 03/06/2021.

for eg :-

```
import sys.
f=open("demofile1.txt","r")
line1=f.readline()
line2=f.readline()
line3=f.readline()
sys.stdout.write(line1)
sys.stdout.write(line2)
sys.stdout.write(line3)
sys.stderr.write("\n No errors occured\n")
f.close()
```

O/P :-

```
Comp.Sc
Students
SecB
No errors occured
```

FILE I/O ATTRIBUTES :-

ATTRIBUTE :

-name :- Returns the name of the file (including path)

-mode :- Returns the mode of the file (r or w etc)

-encoding :- Returns the encoding format of the file or for byte converte

-closed :- Returns true if the file is closed else false

-newlines :- Returns "\r", "\n", "\r\n". None or a tuple

[P.T.O]

containing all the newline types seen :-

eg :-

```
f=open("demofile 1.txt","a+") # 1 open txt file
print (f.closed)
print(f.encoding)
print(f.mode)
print(f.newlines)
print(f.name)
```

for eg :-

```
f=open("demofile 1.txt","r+")
>>> print (f.closed)
false
>>> print(f.encoding)
cp1252
>>> print(f.mode)
r+
>>> print (f.newlines)
None
>>> print (f.name)
demofile 1.txt.
```

04/06/2021

I/O == devices == performs

sys.stdin.read ---> input ---> keyboard

sys.stdout.write ---> output ---> monitor

```
>>> import os
>>> import sys
>>> f=open("demofile 1.txt","r+")
>>> a= sys.stdin.readlines()
hello
>>> sys.stdout.write()
hello
```

Date 05/06/2021

USING +R / R+ MODE :-

- Both are same.
- Both read and write data to a file

eg :-

```
>>> f.read()
'welcome'
>>> f.write('U ALL')
6
>>> f.tell()
13.
>>> f.seek(0)
 0
>>> f.read()
'welcome U ALL'
```

- Rather than overwriting the data the file is appended or added because after read(), the cursor is at end of file and write data from there it self by using write(), so, the data is added.

- If we want to overwrite the data, then we need to perform write (). then read() the data.

eg :-

```
f=open("demofile 1.txt,'r+')
>>> f.write ("Hello everyone")
 14
>>> f.tell()
 14
>>> f.seek(0)
 0
```

[P.T.O]

```
>>> f.read()
'Hello everyone'
```

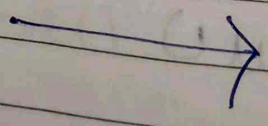
-otherwise after performing read(), we need to set the cursor in the beginning of file by using seek method.

```
f=open("demofile1.txt","r+")
>>> f.read()
'Hello everyone'
>>> f.tell()
14
f.seek(0)
0
>>> f.write("Hiiii \n A")
11
>>> f.seek(0)
0
>>> f.read()
'Hii \n A one'
```

USING W+/+W MODE:-

-Both are same

-when we open a file in w/w+/+w mode the data of the opened file is erased and cursor waits at the beginning of the file to new data to be over written.



```
f=open("demofile1.txt","r")
>>> f.read()
"Welcome vall xii B"
>>> f.close()
>>> f=open("demofile1.txt","w+")
>>> f.tell()
0
>>> f.read()
''
>>> f.write("hello")
5
>>> f.seek(0)
0
>>> f.read()
'hello'
```

CW 04/06/2021

STANDARD I/O AND ERROR STREAMS IN PYTHON:-

- operating system provides package sys including module names as stdin and stdout for inputting and outputting the data into a file.

- Read func" is used for inputting from keyboard and write func" is used for outputting on monitor.

- sys.stdin.read ---> inputting from keyboard
- sys.stdout.write ---> outputting on monitor

EXAMPLE:-

```
>>> import os
>>> import sys
>>> f=open("demofile1.txt","r")
```

[P.T.O]

```
>>> a = sys.stdin.readline()
hello world

>>> sys.stdout.write(a)
hello world
12
```

THE PICKLE MODULE :-

- Python considers everything as an object, so, all data types including list, tuple, dictionary etc. are also considered as objects.

- During execut^n of a program, we may require to store current state of variables so that we can retrive them later to its present state.

EXAMPLE :-

- Suppose you are playing a video game, and after sometime you close it. So, the program should be able to store the current game, including current level/stage, your score, etc as a python object

- To save any object structure along with data, python provides a module called pickle.

- The module pickle is used for serializing and de-serializing any python object structure.

- Pickling is a method of preserving food items by placing them in some sol^n. which increases the shelf life.

→

-In other words, it is a method to store food items for later consumption.

SERIALIZATION:-

-Serialization is the process of transforming data or an object in memory (RAM) to a stream of bytes called byte streams.

-These byte streams in a binary file can then be stored in a disk or in a database or sent through a network.

-Serialization process is also called pickling.

DE-SERIALIZATION:-

-De-Serialization or unpickling is the inverse of pickling process where a byte stream is converted back to python object.

- The pickle module deals with binary file.

-In binary file the data are not written but dumped and similarly, data are not read but loaded.

-The pickle module provides two methods - dump() and load() to work with binary files for pickling and unpickling, respectively.

THE DUMP() METHOD:-

-This method is used to convert (pickling) python object for writing data in a binary file.

[P.T.O]

- The file in which data are to be dumped, needs to be opened in binary write mode (wb).

Syntax of dump() method :-

dump(data_object, file_object)

CW 09/06/2021

where data_obj is the object that have to be dumped to the file.

file_object :- It is a name used to handle a file.

EXAMPLE :-

Program to write record of a student's (roll no, name, gender and marks) in the binary file named as mybinary.data using dump.

PICKLING DATA IN PYTHON PROGRAM :

```
>>> import pickle
>>> listvalues=[1,"Geetika",'F',26]
>>> fileobject=open("my binary.dat","wb")
>>> pickle.dump(listvalues,fileobject)
>>> fileobject.close()
```

- we need to close the file after pickling.

CW 10/06/2021

THE LOAD() METHOD :-

- This method is used to load (unpickling) data from a binary file.

→

- The file to be loaded is opened in binary read (rb) mode.

Syntax of load() is as follows:-

Store_object = load(file_object)

- file_object is the variable used to handle the file which is the pickled python object.

- Store_object is a file used to store the file after loaded.

EXAMPLE:-

- unpickling data in python program

```
import pickle
print("The data that were stored in file are:")
file object = open("mybinary.data", "rb")
objectvar = pickle.load(file object)
fileobject.close()
print(objectvar)
```

O/P:-

The data that were stored in the file are:
[1, 'Geetika' 'F', 26].

CW 17/06/2021

ABSOLUTE PATH:

- It is the full path from the top most directory

eg:-

c:/users/admin/docs/staff.txt.

RELATIVE PATH:

- It is the path to some file with respect to your current working directory.

[P.T.O]

Camlin Page
Date 17/06/2021

eg:-
docs/staff.txt

CW 18/06/2021

SEARCHING IN A TEXT FILE:-

Q.- Suppose "Book.txt" is a file where we will search and count the occurence time of word "is"

Program:-

```
print(str)
L=str.split()
print(L)
count=0
for i in L:
    if i=='is':
        count=count+1
print(count)
fin.close()
```

O/P:-

python is a high level language
it is a case sensitive language
['python', 'is', 'a', 'high', 'level', 'language', 'it', 'is', 'a', 'case', 'sensitive', 'language']
2.

CW 19/06/2021

SEARCHING A STRING IN A TEXT FILE:-

```
fin = open("BOOK.txt", 'r')
str = fin.read()
print(str)
L = str.split()
```

→

Date 19/06/2021

```
print(L)
d=input("enter data to be searched:")
count=0
for i in L:
    if i==d:
        count=count+1
print(count)
fin.close()
```

O/P:-

python is a higher level language
It is a case sensitive language
['python', 'is', 'a', 'higher', 'level', 'language', 'it', 'is', 'a', 'case', 'sensitive', 'language']
-Searching language
2

#count the no. of characters from a file (Don't count white space)

```
fin=open("BOOK.txt",'r')
str=fin.read()
print(str)
L=str.split()
print(L)
count_char=0
for i in L:
    count_char=count_char+len(i)
print(count_char)
fin.close()
```

(P.T.O)

Date 19/06/2021

O/P:- python is a high level language
It is a case sensitive language.
['python', 'is', 'a', 'high', 'level', 'language',
'it', 'is', 'a', 'case', 'sensitive', 'language']
52.

#count the no. of characters from a file.
(with count white space)

```
fin = open("Book.txt", 'r')
str = fin.read()
print(str)
count_char = 0
for ch in str:
    count_char = count_char + 1
print(count_char)
fin.close()
```

O/P:
python is a high level language
It is a case sensitive language
['python', 'is', 'a', 'high', 'level', 'language',
'it', 'is', 'a', 'case', 'sensitive', 'language'].
52.

python is a high level language.
It is a case sensitive language
63

#count the no. of words in a file.

```
fin = open("Book.txt", 'r')
str = fin.read()
```

```
print(str)
L=str.split()
print(L)
count_words=0
for i in L:
    count_words=count_words+1
print(count_words)
fin.close()
```

O/P:

python is a high level language
It is a case sensitive language.

['python', 'is', 'a', 'high', 'level', 'language', 'it', 'is', 'a', 'case', 'sensitive', 'language'].

12.

W 21/06/2021

#count no. of lines in a text file.

```
fin=open("book.txt",'r')
str=fin.readlines()
print(str)
count_line=0
for i in str:
    count_line=count_line+1
print(count_line)
fin.close()
```

O/P:-

['python is a high level language\n', 'It is a case sensitive language'].

2.

count no. of vowels in a text file

```
fin = open("BOOK.txt", 'r')
str = fin.read()
print(str)
count = 0
for i in str:
    if i == 'a' or i == 'e' or i == 'i' or i == 'o' or i == 'u':
        count = count + 1
print(count)
fin.close()
```

O/P:

python is a high level language
It is a case sensitive language
22.

*creating file named as hello.

```
f = open("hello.txt", 'w') # text file.
```

~~f = open("hello.dat")~~

```
import pickle
f = open("hello.dat", 'wb') # binary file.
```

*reading the file.

```
f = open("hello.txt", 'r') # text file / reading.
f.read()
f.readline(n)
f.readlines()
```

(ii) import pickle
f = open("hello.dat", 'rb') #reading binary file.
pickle.load(f)

* writing a file

(i) f = open("hello.txt", 'w') # writing text file
f.write
f.writelines()

(ii) import pickle
f = open("hello.dat", 'wb') #writing binary file.
pickle.dump(f)

(w) 22/06/2021
#reading and writing and searching in a single text file

```
f = open("hello.txt", "w+")
f.write("Hello world")
f.seek(0)
str = f.read()
print(str)
count = 0
for i in str:
    if i == 'l':
        count = count + 1
print(count)
f.close()
```

O/P:-

Hello world
3

(P.T.O)

Date 22/06/2021

#Reading and writing and searching in a single binary file

```
import pickle
f = open ("student.dat", "wb")
l = ['Katherine', 1, 'A']
pickle.dump (l, f)
f = open("student.dat", "rb")
str = pickle.load (f)
print (str)
count = 0
for i in str:
    if i == l:
        count = count + 1
print (count)
f.close()
```

O/P :-

```
['Katherine', 1, 'A']
1
```

writing the data to a file rather than a constant content we may have dynamic contents.

```
f = open ("demofile 1.txt", "w+")
data = input ("Enter msg :")
f.write (data)
f.seek(0)
str = f.read()
print ("my entered data is :", str)
f.close()
```

→

O/P :-

enter msg : hello

msg entered data is : hello

———— x ————

CHAPTER:

Date 24/06/2021

"COMMA SEPARATED VALUE) CSV FILE"

-CSV (comma separated values) is a file format for data storage which looks like a text file. The information is organized with one record on each line and each field is separated by comma.

CSV FILE CHARACTERISTICS:-

- one line for each record.
- comma separated fields.
- space-characters adjacent to commas are ignored.
- field with in-built commas are separated by double quote characters.

WHEN TO USE CSV?

- when data has a strict tabular structure.
- To transfer large database betn programs.
- To import and export data to office applications, Qedoc modules
- To store, manage and modify shopping cart catalogue.

CSV ADVANTAGES:-

- CSV is faster to handle.
- CSV is smaller in size.
- CSV is easy to generate
- CSV is human readable and easy to edit manually
- CSV is simple to implement and parse.
- CSV is processed by almost all existing applicatn.

→

Camlin Page
Date 24/06/2021

CSV DISADVANTAGES:-

- No standard way to represent binary data.
- There is no distinction bet'n text and numeric values.
- poor support of special characters and control characters.
- CSV allows to move most basic data only, complex configuration can't be imported and exported this way.
- problems with importing CSV into SQL (no distinction bet'n null and quotes).

SQL:-

Structured query language is a DBMS (Data Base Management system) software to create, add, modify delete and retrive the data into data base.

NULL:-

- Student contact is a field in student database, which may have Null value as few students may not have any contact details.

25/06/2021

WHAT IS A CSV?

- CSV (comma separated values) is a simple file format used to store tabular data, such as a spreadsheet or database.
- A CSV file stores tabular data (numbers and text) in plain text.

[P.T.O]

- Each line of the file is a data record.
- Each record consists of one or more fields, separated by commas
- The use of the comma as a field separator is the source of the name for this file format

WORKING WITH CSV FILE IN PYTHON:-

① - Import the csv module :-
- csv module is an inbuilt module which can be directly imported through import statement.

```
import csv #importing csv module
```

- Importing csv module is must to use the in-built functions for reading and writing the files.

② - creating and opening csv file :-

Syntax:-

```
with open('filename','w') as csvfile #creating a csv file as writer obj.
```

Eg:-

```
with open('d:\\a.csv','w') as newfile:
with open('myfile.csv','w') as f:
```

Program to write a csv file:

```
import csv
with open("myfile.csv",'w') as newfile:
    furiter = csv.writer(newfile)
    furiter.writerow(['A',1,12])
    furiter.writerow(['B',6,14])
    newfile.close()
```

→

Camlin Page
Date 25/06/2021

writing to a csv file :-

- for writing we open file in 'w' writing mode using open() method which create object like newfile.

ex:-

```
with open("myfile.csv",'w') as newfile.
```

Through csv.writer() method, we create writer object to call writerow() method to write objects.

```
fwriter = csv.writer(newfile)
fwriter.writerow(['A', 1, 13])
fwriter.writerow(['B', 5, 14])
newfile.close()
```

(w) 26/06/2021

- So, to write a csv file we need two methods i.e, writer() and writerow() method

- syntax of writer() method:

```
write_obj = csv.writer(fileobj)
```

- syntax of writerow() method

```
write_obj.writerow(data)
```

READING CSV FILE:-

- for reading the file, we open the file in 'r' mode

ex:-

```
with open("myfile.csv",'r') as newfile
                               fileobject
```

- we need csv.reader() method to read the file and create a read_obj to read each row of the file.

~~Syntax of reader()~~

[P.T.O]

Syntax of reader() method :-
read_obj = csv.reader(file_obj)

-Syntax read_obj
print it directly through for loop.

Program :-

```
import csv
with open("myfile.csv",'r') as newfile:
    nr=csv.reader(newfile)
    for row in nr:
        print(row)
newfile.close()
```

O/P :-

```
['surbhi', '1', '13']
[]
['monak', '2', '13']
[]
```

Program :-

```
import csv
>>> with open("myfile.csv",'w') as newfile:
nw=csv.writer(newfile)
nw.writerow(['surbhi', '1', '13'])
nw.writerow(['monak', '2', '13'])
newfile.close()
17
15
>>>
>>> with open("myfile.csv",'r') as newfile
nr=csv.reader(newfile)
for row in nr:
```

Date 26/06/2021

```
print(row)
newfile.close()
```

```
['surbhi', '1', '3']
['mohan', '3', '13']
```

30/06/2021

ALGORITHM:-

-Step by step instruc^n to do a particular task or to solve a problem

eg :-

-Algorithm for adding two numbers:

STEP1 :- Start.

STEP2 :- Take first number a.

STEP3 :- Take second number b.

STEP4 :- Add a and b and store it in c.

STEP5 :- display c.

STEP6 :- end.

RECURSION:-

-Recursion refers to a programming technique in which a func^n calls itself either directly or indirectly.

[P.T.O]

CW
02/07/2021

Camlin Page
Date 02/07/2021

- Recursion means iteration.

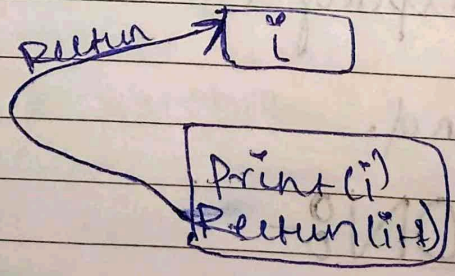
- It means calling a function itself again and again, which is the most powerful tool in programming language.

- It is a way of programming or coding technique, in which a function calls itself for one or more times in its body and returning the return value of this function call procedure

- function calling by itself is done based on a function condition if true it repeats the procedure. Thus, a python recursive function should have a termination condition.

- It can go on forever resulting an infinite loop, so to prevent the function from infinite recursion, we need a termination condition.

- This termination condition is also called base condition.



THE TWO LAWS OF RECURSION:-

- Must have a base case - there must be atleast one base criteria/condition, when such condition meets the function stops calling itself.

Date 02/07/2021

- Must move towards the base case - The recursive calls should move in such a way that each time it comes closer to the base criteria.

02/07/2021

ADVANTAGES OF PYTHON RECURSION

- Reduces unnecessary calling of func^n, thus reduces length of program.

- Very flexible in data structure like stacks, queues, linked list and quick sort.

- Big and complex iterative sol^{n}s are easy and simple with python recursion.

- Algorithms can be defined recursively making it much easier to visualize and prove.

DISADVANTAGES OF PYTHON RECURSION

- Slow.

- Logical but difficult to trace and debug.

- Requires extra storage space. For every recursive calls separate memory is allocated for the variables.

- Recursive func^{n}s often throw a stack overflow exception when processing or operat^{n}s are too large.

[P.T.O]

Camlin Page
Date 03/07/2021

SIMPLE ALGORITHM TO CALCULATE FACTORIAL OF A NUMBER :-

STEP 01 :- Start.

STEP 02 :- Take / Input n, fact, i

STEP 03 : Input value for n from user.

STEP 04 :- Initialize / take fact = 1 and i = 1.

STEP 05 :- Repeat until i <= n
(a) fact = fact * i
(b) i = i + 1.

STEP 06 :- print fact.

STEP 07 :- Stop.

Program :-

```
# factorial of a number.
num = int(input("enter number : "))
fact = 1
for i in range (1, num+1):
fact = fact * i
print ("factorial of number", fact)
```

program to find factorial of a numser using user defined func^n with parameter.

→

Camlin Page
Date 03/07/2021

```
def factorial(num):
    fact = 1
for i in range(1, num+1):
        fact = fact * i
        print(fact)
n = int(input("Enter a no.:"))
factorial(n)
```

(w) 05/07/2021

ALGORITHM TO CALCULATE FACTORIAL USING RECURSION:-

STEP 1:- Start

STEP 2:- Read number n

STEP 3:- call factorial(n)

STEP 4:- Print factorial f

STEP 5:- Stop

factorial(n)

STEP 01:- If n == 1 then return 1

STEP 02:- Else
f = n * factorial(n-1)

STEP 03:- Return f

program to calculate factorial using recursio

[P.T.O]

```
#factorial of a no.
def rec_fact(n):
    if(n==1):
        return(1)
    else:
        return(n*rec_fact(n-1))
#func^n calling
n=int(input("Enter any number:"))
f=rec_fact(n)
print("factorial =",F).
```

~~Enter any number :~~

O/P :-

Enter any number :5
factorial =120.

n*factorial(n-1)
5*factorial(4)
5*4*factorial(3)
5*4*3*factorial(2)
5*4*3*2*factorial(1)
5*4*3*2*1.

#program to display a message:

```
msg=input("Enter a message:")
print(msg).
```

O/P :-

Enter a message : Hi
Hi.

→

program to display a message using user defined func^n.

```
def msg(m):
    print(m)
m = input("Enter msg:")
msg(m)
```

O/P :-

Enter msg : Hi
Hi.

program to display to no. of terms in a fibonacci series.

```
n = int(input("Enter no. of terms in series:"))
n1 = int(input("Enter first no.:"))
n2 = int(input("Enter second no.:"))
i = 1
while (i <= n):
    print(n1)
    n3 = n1 + n2
    n1 = n2
    i = i + 1
```

O/P :-

Enter ~~first~~ no. of terms in series : 5
Enter first no. : 1
Enter second no. : 2
1
2
3
5
8

[P.T.O]

07/07/

ALGORITHM TO CALCULATE SUM OF N NUMBERS:-

Step 01: Start

Step 02: Input the number N

Step 03: Initialize i = 1, sum = 0

Step 04: Repeat until i <= n
sum += i // sum = sum + i
i += 1

Step 05: print sum

Step 06: stop.

ALGORITHM TO CALCULATE SUM OF N NUMBERS USING RECURSION:-

01: Start

02: Input number n

03: call sum (n)

04: Print s

05: stop.

sum (n)
1. if n == 0
return 0

→

Date 07/07/2021.

2. else
   s = n + sum(n-1)
3. return s

#Sum of all the numbers upto n numbers.

```
def rec_sum(n):
    if (n==0):
        return(0)
    else:
        return(n+rec_sum(n-1))
#func^n calling
n=int(input("Enter limit "))
s=rec_sum(n)
print("Sum =",s)
```

O/P :-

Enter limit : 5
Sum = 15

08/07/2021

SIMPLE ALGORITHM TO FIND FIBONACCI SERIES :-

01 : Start

02 : Input number n.

03 : Initialize n1 & n2 value, i = 1

04 : repeat until i <= n
   print(n1)
   n3 = n1 + n2
   n2 = n1

[P.T.O]

```
n3=n2
i+=1
```

05: Stop.

#program to display the terms of a fibonacci series.

```
n1=int(input("Enter 1st no:"))
n2=int(input("Enter 2nd no:"))
n=int(input("Enter no. of terms in series:"))
i=1
while(i<=n)
    print(n1)
    n3=n1+n2
    n1=n2
    n2=n3
    i+=1
```

O/P:-

```
Enter 1st no: 0
Enter 2nd no: 1
Enter no. of terms in series: 5
```

0
1
1
2
3

→

ALGORITHM USING RECURSION FOR DISPLAYING FIBONACCI SERIES:

01: Start

02: input number n

03: if n<=0
print Enter a positive value.

04: Else
repeat i<=n
print f(i)

05: stop.

f(n)
1: if n<=1
return n
2: else
return fib(n-1) + fib(n-2)

```
def fib(n):
    if n<=1:
        return n
    else:
        return (fib(n-1)+fib(n-2))
n=int(input("enter a number"))
if n<=0:
    print("please enter a positive integer")
else:
print("fibonacci sequence:")
for i in range(n):
        print(fib(i))
```

(P.T.O)

09/07/20

Date 09/07/2021

RECURSION ON ARRAYS :-BINARY SEARCH.

ARRAY:-

- Python array are a data structure like list.

- They contain a number of objects that can be of same data types.

- It is used to store similiar type of items.

DIFFERENCE BETWEEN ARRAY AND LIST:-

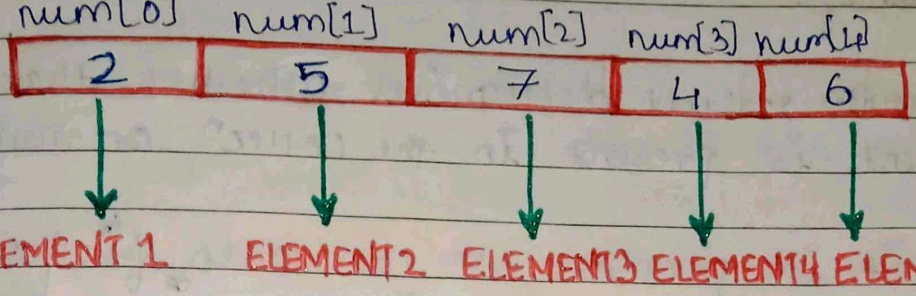
Array	List
-made up of homogenous (similiar) data type.	- made up of heterogenous (different) data type
-can't contain elements of different types.	- can contain elements of different types.
- Smaller memory consumption.	- more memory consumption.
-Better runtime	-Runtime not speedy.
-eg(mohak, Surbhi)	-eg['mohak', 12, B]

- There are two types of Array

→

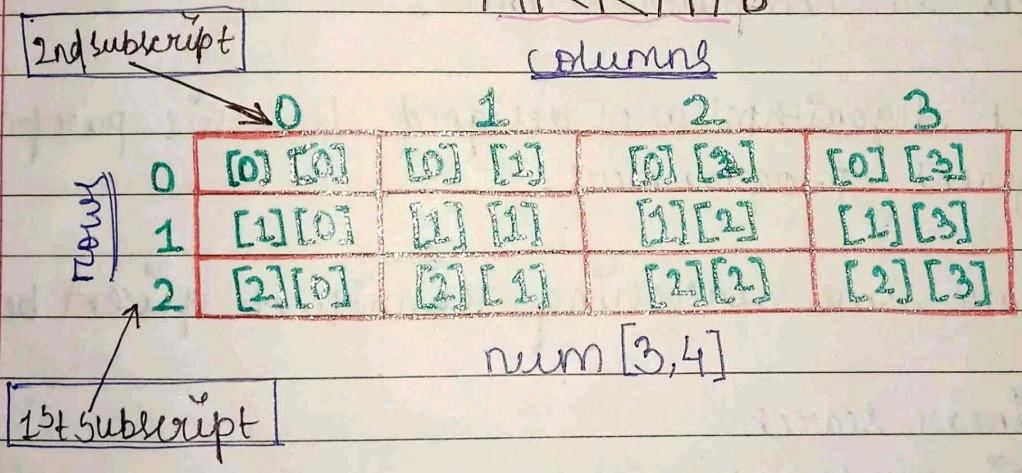
Camlin Page
Date 09/07/2021

01 ONE-DIMENSIONAL ARRAY OR 1-D ARRAY:-



- In above diagram num is an array, it's first element is at 0 index position, next element is at 1 and so on till last element at n-1 index position.

MULTI-DIMENSIONAL ARRAY OR 2-D ARRAY:-



num [3,4]

- In above diagram num is an array of two dimension with 3 rows and 4 columns.

SEARCHING:-

- A computer also stores lots of data to be retrived later as and when demanded by a user or a program.

[P.T.O]

- Searching means locating a particular element in a collection of elements stored in computer memory.

- Search result determines whether that particular element is present in the collection or not.

- If it is present, we can also find out the position of that element in the given collection.

- In computer we store our data by using different data structures like list, array, tuple, queue, stack etc.

- To search and to retrive the required element from the collection of data, we may have different searching techniques in computer science.

- Different Algorithms are designed for this purpose by different programmers.

* These are some searching techniques given below:-

(i) Linear search
(ii) Binary search
(iii) Hashing etc.

LINEAR SEARCH:-

- It is the most fundamental and the simplest search method.

- It is an exhaustive searching technique where →

Camlin Page
Date 10/07/2021

every element of a given list is ~~compared one by one~~ compared with the item to be searched (usually referrid to as 'key').

- So each element in the list is compared one by one with the key.

- This process continues until an element matching the key is found and we declare that the search is successful.

- If no element Matches the key and we have traversed the entire list, we declare the search is unsuccessful i.e, the key is not present in the list.

- This item by item comparision is done in the order in which the elements are present in the list, beginning at the first element of the list and moving towards the last.

- Thus, it is also called sequential or serial search. This technique is ~~usefull~~ useful for collection of items that are small in size and are unordered.

24/07/2021

BINARY SEARCH:-

- The binary search is a search technique that is used for the ordered elements in the list to quickly search a key.

- for numeric values, the elements in list may be arranged either in ascending or descending order of their key values.

[P.T.O]

- for textual data, it maybe arranged alphabetically starting from a to z or from z to a.

- In binary search, the key to be searched is compared with the element in the middle of the sorted list. This could result in either of the three possibilities.

(i) The element at the middle posⁿ itself matches the key.
(ii) The element at the middle posⁿ is greater than the key
(iii) The element at the middle posⁿ is smaller than the key.

- If the element at the middle posⁿ matches the key, we declare the search successful and the searching process ends.

(10) 29/07/2021

ALGORITHM OF BINARY SEARCH

- Binary search includes (numlist, key)

Step 01:- SET first = 0, last n-1.

2:- calculate mid = first + last / 2.

3:- while first <= last repeat step4.

4:- if numlist(mid) = key
print "Element found at posⁿ"
"mid+1
STOP

Camlin Page
Date 22/07/2021

ELSE
if numlist(mid) > key. THEN last
=mid - 1
ELSE first = mid + 1.

5 :- PRINT "search unsuccessful"

BINARY SEARCH USING RECURSION:-

Program :-

```
def binary search (arr, first, last, x):
    if last >= first:
        mid = int((first+(last - first)/2)
        if arr(mid) == x:
            return mid
        elif arr(mid) > x:
            return binary search (arr, first, mid-1, x)
        else:
            return binarysearch (arr, mid+1, last, x)
    else:
        return -1.
arr = [1, 3, 5, 6, 7, 8, 10, 13, 14]
X = 10
result = binarysearch
```

24/07/21

APPLICATIONS OF BINARY SEARCH :-

- Binary search has numerous applications including searching a dictionary or a telephone directory, finding the element with minimum value or minimum value in a sorted list, etc.
- modified binary search techniques have far reaching applications such as indexing

Date 24 / 07 / 21

in databases, implementing routing tables in routers, data compression code, etc.

eg:-

Consider a sorted list comprising of 15 elements:

numlist - [2, 3, 5, 7, 10, 11, 12, 17, 19, 23, 29, 31, 37, 41, 43]

we need to search for the key, say 17 in numlist. The first, middle and last element identified in numlist along with their index values shown below:-

Elements in sorted numlist along with their index value:-

	FIRST					MID					LAST		
Index in numlist	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
value	2	3	5	7	10	11	12	17	19	23	29	31	37	41	43

working of binary search using steps given in Algorithm:-

	FIRST	LAST	MID	numlist[mid] == K	Key < Lmid	first <= last
At start	0	14	(0+14)/2 = 7	not known	not known	0 <= 14 true
Iteration	0	14	7	[17 == 17] yes	key found search terminates	