PYTHON PROGRAMMING

FEAUTURES:-

1. Easy to read and learn
2. Portable
3. Scalable
4. Interactive Mode
5. GUI Programming
6. DB Support
7. Library
8. Opensource
9. Platform Independant
10. Interpret
11. Extensible
12. Embeddable
13. Multiparadigm
14. Object oriented

Interactive and script mode.

Addition of 2 numbers:

```
num1 = 6.5
num2 = 5.5

Sum = num1 + num2
print ("The sum is", sum)
```

Square root operations:-

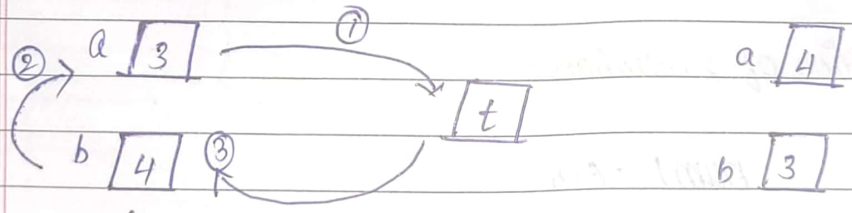
```
num = 8
num_sqrt = num ** 0.5
print ('The squareroot is', num_sqrt)
```

program to calculate area of triangle:

```
a=5
b=6
c=7

s = (a+b+c)/2
area = s*(s-a)*(s-b)*(s-c)**0.5
print ('The area of triangle', area)
```

Swapping 2 nos:-



command statement

```
# This is swapping program.
a=3
b=4
t=a
a=b
b=t
print ('After swap', a, b)  (or)
```

→ string format → integer value conversion.

```
a = int(input('Enter the first number'))
b = int(input('Enter the second number'))
t = a
a = b
b = t
print('After swap', a, b)
```

Program to convert celcius to Farenheit.

$$f = x \times 1.8 + 32$$

Program:-

```
c = 30
f = (c * 1.8) + 32
print('Farenheit value', f)
```

if input is to be given by user.

```
c = int(input('Enter celcius value'))
```

- [not display]

Command statement

```
# c = 30
f = (c * 1.8) + 32
print('Farenheit value', f)
```

VARIABLES:-

Variables are the place where certain values are stored
Assignment operators are used to assign value.

IDENTIFIERS:-

Identifier is a name which is used to identify a variable, function or a class.

Rules

- It starts with capital letter (A-Z) (or) small letter (a-z) (or) can start with underscore.
- It can be followed by 1 or more letters, (_) or digits.
- No special characters such as dollar, percentage (or) at [$, %, @]
- Identifiers are case sensitive.
  num 1 <-> Num 1
- No key words are used for identifiers.
- Class name always starts with upper case. All other identifiers shall start with lower case.

[Keywords - words that already have meaning]

OPERATORS:-

Specially designated symbols used to perform certain arithmetic (or) any operation.

EXPONENT OPERATOR :- **

RELATIONAL OPERATOR :- Comparing 2 things.
True / False - Boolean → capital letters reserved.

* >, <, >=, <=, ==, !=

ASSIGNMENT OPERATOR

Equal to (=) , (+=) , (-=) , (*=) , (/=) , (/-=)

a = 5
a += 3
a = 8

BITWISE OPERATOR

& - And | - or

LOGICAL OPERATOR :-

Logical and & Logical not

MEMBERSHIP OPERATOR

Keyword : in , not in

To check the given element is present in the sequence (or) not.
Yes - in
No - not in

a=60
b=13
c=0

Binary value

60 = 0011 1100
13 = 0000 1101

c = a & b
print(c) (12)

AND = 0000 1100
OR = 0011 1101

c = a | b
prin(c) (61)

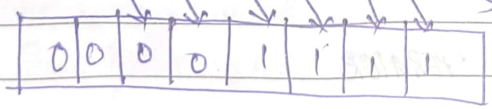
While applying XOR

13 0000 1101
60 0011 1100
0011 0001 - (49)

RIGHT SHIFT OPERATION (divided)

By 2 :- 15>>2

60 0 0 1 1 1 1 0 0



= (15) //

By 1:- 30>>1

60 0 0 0 1 1 1 1 0

= (30) //

60 → 30>>1 , 15>>2 //

LEFT SHIFT OPERATION :- (multiplied)

BY 1:-

60 0 0 1 1 1 1 0 0

0 1 1 1 1 0 0 0 ①

120 << 1

BY 2 :-

60 = 1 1 1 1 0 0 0 0 ← from eq①

240 << 2

```
c = ~a
print (c)
c = a^b
print(c)

c = a>>2
print(c)
```

BITWISE OPERATION :-

a = 10
b = 4

(AND)

```
1010
0100
1110  - 14
```

```
c = a & b
print (c)
```

IDENTITY OPERATOR

↳ If two variables have same memory location 'is' operator return true.

```
a1 = 5
b1 = 5
a2 = 'Hello'
b2 = 'Hello'
a3 = [1,2,3]
b3 = [1,2,3]
```

} as numbers can be added or removed ~~interpreter can't~~ interpr...

Output

```
print (a1 is not b1)   | False
print (a2 is b2)       | True
print (a3 is b3)       | False
```

If `c3 = a3`

```
print (c3 is a3)   - True
```

MEMBERSHIP OPERATOR. (in not in)

Used to find whether the particular value is in sequence or not.

```
x = 'Hello World'                    Output

print ('H' in x)                     True
print ('Hello' in x)                 True
print ('Bello' in x)                 False
```

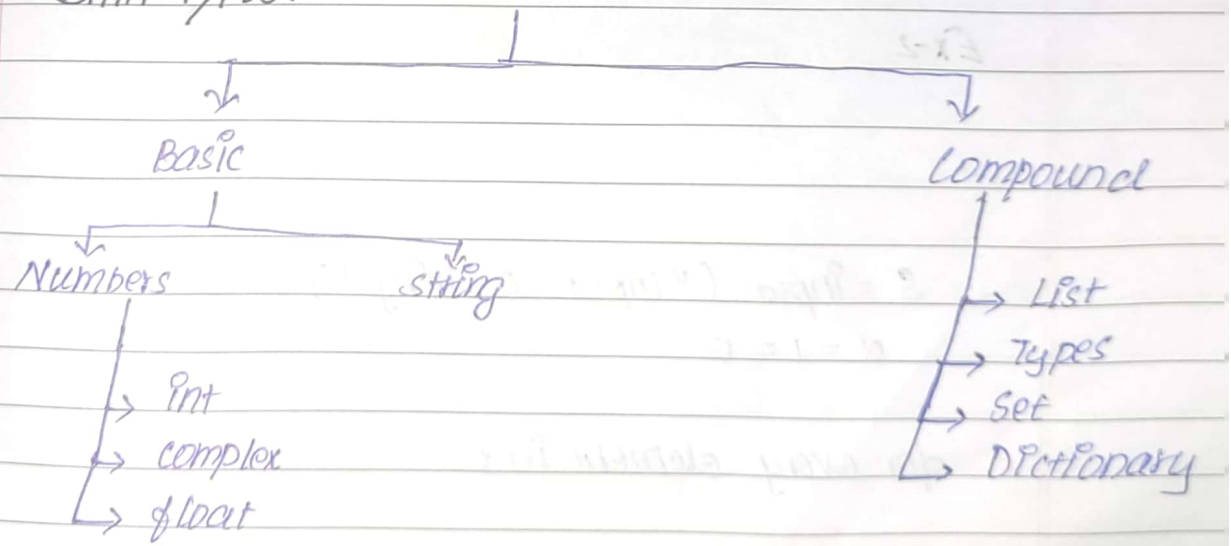
```
y = { 1 : 'a' , 2 : 'b' }

Key   ← print (1 in y)               True
Value ← print ('a' in y)             False
```

∴ Keys can only be printed, values can't.

*

marks DATA TYPES:-



List - • Heterogenous collection of element
[ ] • Mutable - can change the value
• Orderly manner

Tuple:- ( )

* Order
* Heterogenous collection
* Not mutuable.

Dictionary

* Not ordered
* Heterogenous collection
* Mutuable
* Key should be unique.

EX-1

Largest Element in List.

```
list1 = [10, 20, 4, 5, 9]
print ("Largest Element is "; max (list1))
```

EX-2

```
s = input ("Input a string")
d = l = 0

for every element in s

        s = text 5

if c.isdigit();
        d = d + 1
```

```
elif
        c.isalpha()
        l = l+1
else
        pass

print("Letters"; l)
print("Digit"; d)
```

Ex-3 :-

```
my_dict = {'Brand': 'Audi'
           'Segment': 'SUV'
           'Price': 90000
           'colour': 'Black'}

for i in my_dict:

    print(i, mydict[i])
```

MODULE - 2

LOOPING CONSTRUCTS:-
↳ iterating process

FOR LOOP:-

```
numbers = [6,5,3,8,4,2,5,4,11]
sum = 0
for val in numbers
    sum = sum + val
print('The sum is', sum)
```

inside loop →

[val ↳ values present in array]

Output

48

The sum is 48.

sum = 6 → sum = 6+5 ..... , sum = 6+5+3 ..

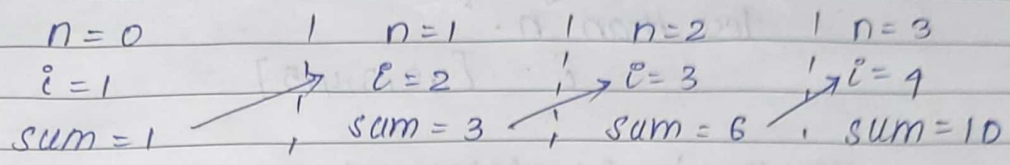
After the values are added completely it moves to print statement and the output is printed.

WHILE LOOP:-

```
n = 4
sum = 0
i = 1
while i <= n
    sum = sum + i
    i = i + 1
print("The sum is", sum)
```

Output :-

The sum is 10



PERFECT NOS. PROGRAM.

```
n = int(input("Enter the number"))
sum1 = 0
for i in range(1,n):
    if(n%i==0):
        sum1 = sum1 + i

if(sum1==n):
    print("The number is perfect")
else:
    print("The number is not perfect")
```

Perfect number is a number whose divisors when added gives the number itself.

For eg :-

6 divisible by 1, 2, 3.

6 = 1 + 2 + 3

6 = 6 //

For example:-

If we take $n = 6$

so the range of i is one number less than n.

$i = [1,2,3,4,5]$

Now the 'for i' statement gets executed

```
if (n % i == 0)
✓   6 % 1 == 0      1   (1+2)
✓   6 % 2 == 0      3   (3+3)
✓   6 % 3 == 0      6
x   6 % 4 == 0
x   6 % 5 == 0
```

Now the program gets to next statement

```
if (sum1 == n)        1+3+2 = 6,
```

print('The number is perfect') is printed.

FACTORIAL

```
n = int(input("Enter the number"))
n=3          fact = 1
             while (n>0):
1x3=3            fact = fact * n
(n-1)=2          n = n-1
3x2          print("Factorial is")
(2-1) 3x2x1  print(fact)
                          Factorial is 6
```

LOOP CONTROL STATEMENTS :- → break, → continue, → pass

BREAK:-

It terminates the loop statement and transfers the execution to the statement immediately following the loop.

CONTINUE:-

It is used to skip the rest of the code inside the loop for the current iteration only. The loop doesn't terminate but continues to the next iteration.

PASS:-

It is a null statement. It is used when a statement is required syntatically but you do not want any command or code to execute. Thus it results in no operation.

Eg:-

```
for val in 'Python'
    print(val)
print('The end')
```

output

P
y
t
h
o
n

Break :-

```
for i in 'Python'
    if i == 't'
        break
    print (i)
print ('The end')
```

output :-

P
y
The end

Continue

```
for i in 'Python'
    if i == 't'
        continue
    print (i)
print ('The end')
```

output :-

P
y
h
o
n
The end.

FUNCTION :-

- Defn - where we decide what the function should do.
- call - calling the same function in the program again.
- return

Syntax :-

```
def function_name (parameter):      (Defn)
    ____
    ____

function_name ( )                   (call)
add()                               (Operation)
```

Ex:-

```
def add():
    a = int(input("Enter 1st no"))
    b = int(input("Enter 2nd no"))
    c = a+b
    print(c)
print('The sum is', c)
add().
```

Eg:-

With parameters:-

```
def x(name):
    print('Welcome' + name)
x('John')
```

output :- Welcome John.

Eg:-

```
def y():
global x
    x = 4000

x = 100
y()
print(x)
```

→ This statement makes the 'x' in the function global and changes the output value, if not applied global it will be local variable and looses its value when called in main fn.

RECURSSION:-

A function calling itself again & again

Eg:-

```
def factorial (x) :
    if x == 1 :
        return 1
    else :
        return (x * factorial (x-1))

num = 4
print ("The factorial is")
print (factorial (num))
```

Output:-

= [4 * factorial (3)]

= [4 * 3 * factorial (2)]

= 4 * 3 * 2 * factorial (1)

= 4 * 3 * 2 * 1 = 24 //

TYPES OF ARGUMENTS :-

1. REQUIRED ARGUMENTS :-

```
def printme (str):
    print (str)
    return
printme ()
printme ("Hello")
```

Absence of argument leads to an error in a function.

(function call.)

2. KEYWORD ARGUMENTS :-

```
def printinfo (name, age):
    print (name)
    print (age)
    return
printinfo (age = 50, name : 'xyz')
printinfo (name = 'abc', age = 55)
```

If values are only passed exchanging of age eg number is not possible whereas while expressing it in a keyword form it can be exchanged.

For eg:-

age = 50 name : 'xyx' ✓

whereas printinfo (name, age) ✗
printinfo (50, 'xyx')

will not work under this case as name cannot be '50' or age cannot be 'xyx'.

DEFAULT:-

```
def printinfo(name, age = 50)
    print(name)
    print(age)
    return
printinfo(name = 'lmn')
printinfo(name = 'xyz', age = 90)
```

o/p

lmn 50
xyz 90

VARIABLE LENGTH / (*)

```
def printinfo(arg1, *vt)
    print(arg1)
    for i in vt:
        print(i)
    return
printinfo(10)
printinfo(10,20,30,40)
```

o/p

ANONYMOUS FUNCTIONS

```
sum = lambda arg1, arg2 : arg1 + arg2
print(sum(4,10))
```

FIBONACCI :-

```
def fibonacci(n):
    if (n<=1):
        return n
    else:
        return fibonacci(n-1) + fibonacci(n-2)
n = int(input("Enter the number of terms"))
for i in range(n):
    print(fibonacci(i))
```

start //

Output:-

⇒ Enter the number of terms = 5 {0,1,2,3,4}

understanding

f(0) = 0 //
f(1) = 1 //
f(2) = f(1) + f(0) = 1+0 = 1 //
f(3) = f(2) + f(1) = 1+1 = 2 //
f(4) = f(3) + f(2) = 2+1 = 3 //

⇒ 0 1 1 2 3 //

GCD of TWO NUMBERS :-

```
num1 = int(input("Enter the first no"))
num2 = int(input("Enter second no"))
gcd = 1
for i in range(1, min(num1, num2)):
    if num1%i==0 and num2%i==0:
        gcd = i
print("GCD is", gcd)
```

Output:-

num1 = 12
num2 = 18 gcd : 6

{#so the value of i will be 1 to 11.}
Now from 1 to 11 all the values will be checked through iteration and the last minimum value that divides both the number in between 1 to 11 is the gcd value of the given numbers}.

MULTIPLICATION TABLE:-

```
num = int(input("Display multiplication
                table of "))
for i in range(1,11):
    print(num,'x',i,'=',num*i)
```

Output:-

num = 4
4x1 = 4
4x2 = 8
4x3 = 12
4x4 = 16
4x5 = 20
.
.
.
.
4x10 = 40.