Python

→ python is a general purpose, high level, interpreted, case sensitive, dynamically typed programming language.

→ python supports multiple language patterns like,

i) structured oriented
ii) functional "
iii) object oriented.

→ python was developed by Guido van Rossum. in 1989 and released in 1991.

→ Using python we can develop different types of applications.

i) Console app's
ii) Desktop GUI
iii) Web
iv) network program
v) Data analysis
vi) machine learning
vii) Artificial Intelligence
viii) Business App's
ix) Audio & Video Apps
x) Gaming App's.

* features of python:

- → easy to learn and use
- → simple syntax
- → expressive language
- → object oriented support
- → GUI support
- → fresh and open source
- → cross platform
- → huge standard library
- → extensibility
- → portable.

* Versions of python:

- → python 0.9.0.
- → python 1.0
- → python 2.0
- → python 3.0
- → python 3.8.1 in dec-2019

* steps to download and install python software:

- → Go to google type download python.
- → click on link www.python.org
- → click on download python 3.8.1
- → download .exe file, double click on .exe file
- → activate the check box Add python 3.8 to path.
- → install next next next Enter.
  click on

Different ways to write python Code and Execute :-

→ using interactive mode

→ using script mode

→ using python IDLE

→ using pycharm Editor.

→ interactive mode :-

→ In interactive mode as soon as when we write Code, it will Execute

→ Go to search type pytha, click on IDLE

Ex: a = 10
b = 20
a+b = 30.

→ Script mode :-

→ In this mode we have to write your python script and save with (.py)

→ open notepad, print('welcome to durga soft').

→ click on file menu, click on save, give file name test.py, save in desktop.

→ To run test.py file through Command prompt.

→ Go to Command prompt type cd — desktop ↳ Enter (4)

→ python — test.py ↳ enter

→ To run test.py using python IDLE.

→ Go to python IDLE click on file menu and open, choose test.py file from desktop, click on open → Go to run menu, click on run module.

✱ Datatypes:

set:

i) set is unordered collection of unique elements

ii) set can be create by using " { } "

iii) set will allow different datatype elements

→ set is mutable, and modify.

→ To create an empty set we use 'set()' function

→ It cannot accept duplicates

Ex: s = {10, 'a', "durga", 34.5, 10}

print(s) o/p: durga, 'a', 10, 34.5

print(type(s))

empty set:

s = set()

print(s).

* dictionary:

→ It is unordered Collection of elements
→ In dictionary element should be ~~key~~ pair of "key: value"
→ In dictionary keys are immutable and unique
→ In dictionary Values are mutable and need not be unique
→ Dictionary Can be create by using { }
→ In dictionary keys and values Can be of any datatype

ex: d = {1: 'sai', 1: 'durga', 2: 'sai', 2.2: 'a', b: bat}
print(d)

o/p: 1: sai, 2: sai, 2.2 = a, b = bat.

print (d.keys()) = only keys
print (d.values()) = only values
print (d.items()) = both keys and values.

* Range

→ It is used to generate sequence of numbers
→ By default range starts from zero
→ Range is immutable, once we create values we can't change

```
r = range(10)
Print(list(r))
Print(list(range(0,10))) - 0 ——— 9
Print(list(range(2,10))) - 2 ——— 9
Print(list(range(2,10,2))) — 2, 4, 6, 8.
```

* operators in python:

→ operator is a symbol which is used to perform required operation

i) Arithmatic operators = +, −, *, /, //, **, %.
(/ → quoient, // → integer value, % → remainder)

ii) Relational = <, >, ==, <=, >=, !=

iii) Assignment = =, +=, −=, *=, /=, //=, %=, **=
(= → assign, += → add and assign, −= → subract and assign.)

iv) Logical = AND, OR, NOT

v) membership = in, not in

vi) Identity = is, not is

vii) bitwise

* program for assignment operators:

```
a = 10
Print(a += 2) = 7
Print(a -= 2) = 5
Print(a *= 2) = 10
```

```
Print(a /= 2) = 5.0
Print(a //= 2) = 2.0
Print(a %= 2) = 0.0
```

Logical : AND = Both are true else false

OR = one is true, true else false

NOT : True = Cond is false

False = Cond is true.

Ex: Print(2<3 and 3>4) = False

Print(2<3 or 3>4) = True

Print((not (2<3)) = False because NOT is there.

* membership :

→ These operator are used to check, whether given value is present in sequence(or) not

in = True → value present in sequence else false

not in = True → not present in sequence else false

Ex: lst = [2, 3, 4, 0, 6, "Sai", 'a']

Print(100 in lst) = False

Print(2 in lst) = True

Print(100 not in lst) = True

Print(3 not in lst) = False

Print('a' in lst) = True.

* Identity:

→ These operators are used to check whether / Variable are having same identity (or) different identity.

is = True, if both Variable having same identity else false

is not = True, if both Variables not having same identity else false

Ex:

```
a = 2
b = 2
Print(a is b) = True
Print(a is not b) = False
```

```
a = 2
b = 3
Print(a is b) = False
Print(a is not b) = True
```

* id() function:

→ It is a built-in function is used to get memory address of the object

Ex:

```
a = 2
b = 3
Print(id(a))
Print(id(b))
Print(id(a) == id(b))
```

```
a = 23
b = 23
Print(id(a))
Print(id(b))
Print(id(a) == id(b))
```

a = 8
b = 11

* bitwise operators:

1. bitwise AND (&)
2. " OR (|)
3. " XOR (^)
4. " NOT (~)
5. " left shift (<<)
6. " Right shift (>>)

2^3	2^2	2^1	2^0
8	4	2	1
a = 8 = 1	0	0	0
a = 11 = 1	0	1	1
bin(a & b) = 1	0	0	0

$2^3 + 2^2 + 2^1 + 2^0$

(a & b) 8 0 0 0 = 8

bin(a | b) = 1 0 1 1

$2^3 + 2^2 + 2^1 + 2^0$

(a | b) 8 0 1 1 = 11

bin(a ^ b) = 0 0 1 1

$2^3 + 2^2 + 2^1 + 2^0$

0 0 2 1

= 3

* Truth table:

a	b	a & b	a | b	a ^ b	~a
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

Sol: a = 8
b = 11

Print bin(a & b) = 8

Print (a | b) = 11

Print (a ^ b) = 3

Print (~a) = -9

Print (~b) = -12

(~a)

a = -(a+1)

a = 8 = -(8+1)

= -9

(~b) = b = -(b+1)

= - b = -12.

* left shift & right shift:

Left shift means no. of bits move to left hand

Right " " " " " move to right hand.

Ex: a = 8

print(a<<2) = 32.

print(a>>2) = 2

2^5	2^4	2^3	2^2	2^1	2^0
32	16	8	4	2	1
a=8		1	0	0	0
b=11				1	1

(a<<2)

1	0	0	0		

32+0+0+0 = 32

(a>>2)

1	0	0	

2+0+0+0

= 2

Control statements

→ Control statements are used to maintain the control flow of our program execution

→ There are of three types:

1. Conditional = if, ifelse, nestedif, elif
2. Iterative = for, while
3. Transfers = pass, break, continue

i) if statement:

→ It is used to test specific condition, if condition is "true" then if block will execute otherwise nothing will execute:

Syntax

```
if Condition :
    statement -1
    statement -2
        }
    statement -n.
```

Ex:

```
a = 100
if a==100:
    print("true")
```

(ii) if else:

→ else block will execute only, when the if condition is false

Syntax:

```
if Condition :
    ========
    ========
else :
    ========
    ========
```

Ex:

```
a=10
if a==100:
    Print("true")
else
    print("false").
```

* C.A.P to check even (or) odd number?

```
a = int(input("enter number"))
if a % 2 == 0:
    Print(a, "even number")
else:
    Print(a, "odd number")
```

iii) Nested if:

→ An if block (or) else block which is having one more if-else is called nested if.

→ Nested if is used to test multiple conditions.

Syntax & program:

```
a = 10
if a > 20:
    Print("Cond1 is true")
else:
    if a > 15:
        print("Cond2 is true")
    else:
        print("all conditions are false")
```

iv) elif :

→ When we have multiple if Conditions, in general if one of the 'if' Condition is true, other 'if' Conditions need not be test

→ let us Consider the programs

```
x = int(input("Enter number"))
if x==1:
    print('one')
if x== 2:
    print("two")
if x== 3:
    print("three")
if x==4:
    print("four")
else:
    print("Invalid number")
```

```
x = int(input("Enter number"))
if x==1:
    print("one")
elif x==2:
    print("two")
elif x==3:
    print("three")
elif x==4:
    print("four")
else:
    print("Invalid number")
```

→ In the above Example, when we enter input as "one", o/p is one and Invalid number.

→ Because in above Example if one of the 'if' Condition is true, still other if Conditions are test. / to avoid this we use "elif".

* for loop:

→ It is used to iterate the elements of Collection (or) sequence what the order they appears

Syntax: for Variable in sequence:

    Statements - 1

    ---
    statements - n.

Ex: lst = [2, 'a', 34.5, 'sai']
print (lst)
for i in lst:
    print(i, type(i)).

o/p:
2
9
34.5
sai

Ex: nested for

numlist = [1, 2, 3, 4]
charlist = ['a', 'b', 'c']
for n in numlist:
    print(n)
    for c in charlist:
        print(c)

* Nested for loop:

A for loop which is having one more for loop with in it is called nested for loop.

→ If nested for loop first loop is outerloop and second loop as inner loop

* * → For every iteration of outerloop, inner loop should finish it all iterations

* while loop:

→ It is used to execute no. of statements till the condition is true, once the condition is false, then control will jump out of the loop.

Syntax: while condition:
    statements - 1
    statement - n

Ex:-
```
i = 0
while i<=10:
    print(i)
    i = i+1 (or) i+1 = 1
```

o/p:
```
0
1
2
!
!
10
```

Ex:- Break:
```
i = 0
while i<=100:
    print(i, end=" ")
    if i == 50:
        break
    i = i+1 (or) i+=1
```

→ Note: When we have fixed no. of iterations (or) limited iterations then we use for loop

→ When we have large no. of iterations (or) unlimited iterations then we use while loop.

* break:-

→ It is used to stop the iteration, though the condition is 'true'.

Ex: break using for loop

1 → 20

```
for i in range(50):
    if i == 20:
        break
    print(i, end=" ")
```

Continue :-

→ It is used to skip a current iteration and continue with next iteration

Ex:

```
for i in range(50):
    if i == 20:
        continue
    print(i, end=" ")
```

} 20 don't print and 21 will start to print upto 50.

Pass: It is a keyword, it can be used to make a block of code as empty

Ex: 1

```
for i in range(50):
    pass
```

Ex: 2

```
if a = 100
    if a == 100:
        pass
    else:
        Pass
```

* join of string:

join something to string

* string reverse:

-> using join function we can reverse the string

-> we can also use string slicing.

reverse:

```
print("".join(reversed('SAP')))
```

join:

```
print(":".join("python"))
```

* string split :-

```
s="python is very easy"
print(s)
s1=s.split(" ")
print(s1)
```

Ascending to descending order

* To get Complete functions :-

```
print(dir(str))
```

* To get Complete info :-

```
help(str)
```

* strip():

It returns a copy of the string with both leading and trailing characters removed

Ex: s="adurga"

```
print(s.strip('a'))
```

o/p: durg

* lstrip:-

It returns a copy of the string with leading characters only

Ex: s="adurga"

```
print(s.lstrip("ad"))
```

o/p: urga.

* rstrip():

It returns only the copy of the string by the trailing character only

Ex:
s = "adurga"
Print(s.rstrip("g,a"))
o/p: adur

* Find():

used to find the substring from given string / returns '-1' if there is no substring.

Ex:
s = ("python is very easy")
Print (s.find("is"))
o/p: 7

Print (s.find("x"))
-1

* MAX():

highest Alphabetical character in string

* MIN():

lowest Alphabhetical character in string

Ex: s = "durgasoft"
Print (max(s))
Print (min(s))
o/p: u by ASCII values
a

* index():

index of substring

* rindex():

highest index of substring

Ex: s = "python is very easy and it is pop and it is interpreter"
Print (s.index("is"))
Print (s.rindex("is"))

* partition :

split the string at the first occurrence of seperator and returns a tuple

Ex:- s = "python is very easy and it is wp"

print (s.partition("is"))

o/p:- ('python ', 'is', ' very easy and it is wp')

* startswith() :-

returns True if the suffix is ~~ends~~ starts (or) else False

* endswith() :-

returns True if the suffix ends (or) else False

Ex:- s = "Anas"

print (s.startswith('A'))

print (s.endswith('s'))

* isdigit : (19)

returns "true" if all are digits (or) else False

* isalpha :

returning "true" if all are alphabets (or) else false

* isalnum :

returns "true" if it contains both alphabets and digits (or) else false

Ex:- s = ('123')

print (s.isdigit())

o/p :- True

Ex:- s = ('abc')

print (s.isalpha())

o/p :- True

Ex:- s = ('abc123')

print (s.alnum())

o/p :- True..

* W.A.P → print table:

```
n = int(input("Enter a number"))
for i in range(1, 11):
    print(n, '*', i, "=", i*n)
```

* list* is in backside of the book

Tuple

→ tuple membership test:

```
tpl = (a, p, p, l, e)
print('a' in tpl) = True
```

in, not in

```
print('d' in tpl) = False
print('d' not in tpl) = True.
```

→ print(len(tpl))
= length of tuple

→ print(max(tpl))
= max element in tpl by ASCII

→ print(min(tpl))
= min-element in tpl by ASCII

→ print(sum(tpl))
= Sum of elements in tpl.

* Convert string into tuple:-

```
str = "durgasoft"
print(str)
tpl = tuple(str)
print(tpl).
```

o/p:- ('d', 'u', 'r', 'g', 'a', 's', 'o', 'f', 't')

* Create one item in tuple put Comma (,) after that item

* Convert list into tuple:

```
lst=[2, 3, 4, 5]
Print (lst)
tpl = tuple (lst)
Print (tpl)
```

o/p:= ('2', '3', '4', '5')

* Convert tuple to string:

```
tpl = (2, 3, 4, 5)
Print (tpl)
str = string (tpl)
print (str)
```

o/p: (sai,)

Set (2)

* Creating empty set:

```
s = set ()
Print (s).
```

→ It is Completely unordered.
→ It Cannot accept duplicates
→ It is mutable (or modify.

* adding element to set:

```
s = {2, 'A', 2.3, '6', 'sai'}
s.add('d')
Print (s).
```

⇒ It add only one item

→ It Cannot add multiple items.

* using "update" method we can add multiple items into set

```
s.update(['d', 3, 5, 6])
Print (s).
```

⇒ update is add more elements

* discard element from set :-

S = {2, 'A', 2.3, 6, 'sai'}

S.discard('A')

Print (S)

o/p: {2, 2.3, 6, sai}

→ If you can discard the item which is not in set nothing will change in past set.

→ S.remove('A')

Print (S)

o/p: {2, 2.3, 6, sai}

→ If you can remove the item which is not in set the error is occured.

* Set operations :-

1) Union (|) :-

A = {1, 2, 3, 4, 5}

B = {4, 5, 6, 7, 8}

Print (A | B)

o/p: {1, 2, 3, 4, 5, 6, 7, 8}

Print (A.union(B))

ii) Intersection (&) :-

A = {1, 2, 3, 4, 5}

B = {4, 5, 6, 7, 8}

Print (A & B)

(or)

Print (A.intersection(B))

o/p: {4, 5}

iii) Difference (-) :-

A = {1, 2, 3, 4, 5}

B = {4, 5, 6, 7, 8}

(A-B) = not in B

(B-A) = not in A

Print (A-B) = {1, 2, 3}

Print (B-A) = {8, 7, 6}

iv) Symmetric difference (^) :-

A = {1, 2, 3, 4, 5}

B = {4, 5, 6, 7, 8}

Print (A ^ B)

o/p: {1, 2, 3, 6, 7, 8}

* membership item in set :

A = {1, 2, 3, 4, 5}

Print (1 in A) : True

Print (1 not in A) : False

Print (6 in A) = False.

* length of set :

A = {1, 2, 3, 4, 5}

Print (len(A))

o/p : 5

* max of set :

A = {1, 2, 3, 4, 5}

Print (max(A))

o/p : 5

* min element of set :

A = {1, 2, 3, 4, 5}

Print (min(A))

o/p : 1

* Sum of element in set :

A = {1, 2, 3, 4, 5} (15)

Print (sum(A))

o/p : 15

Print (sum(A, 100))

o/p : 115

→ clear method is used to make the set as empty.

→ To delete entire the set we use "del" keyword.

S = {1, 2, 3, 4, 5}	S = {1, 2, 3, 4, 5}
S.clear()	del S
Print (S)	Print (S).
	~

Dictionary

Key : value

Key → Immutable, unique → Any data type

value → mutable, duplicates are also allowed → Any data type

→ It is unordered collection of key and values

Accessing elements from dictionary

~~d[1] : "sai"~~

→ To access value "key" must be required

→ we can pass the "key" in two ways:-

print(d[1]) — 1st

print(d.get(1)) — 2nd

→ If the key is present both options will work fine, if not present the error is occured — 1st

→ get functions will returns "none"

d = {1: "sai", 2: "durga", 3: "durga", 'a': 'apple', 6: 33}

print(d)

print(d[1]) ⇒ key error.

print(d.get(3))

print(d.get(9, "not found"))

change (or) add elements

→ To modify the value "key" is required.

→ If the "key" is present in dictionary value gets update

→ If the "key" is not present in dictionary The "key: value" pair is added to dictionary.

```
d={"empid": 888, "empname": "durga"}
print (d)
d["ename"]="mohan"
print (d)
d["age"] = 33
print(d)
```

o/p: {empid: 888, empname: Mohan
age = 33}

* delete (or) remove elements :-

```
d={1: "sai", 2: "durga", 'a': apple,
   'b': 'ball'}
del d[1]
print (d)
```

o/p :- {2: "durga", 'a': "apple", 'b': "ball"}.

```
del d
print (d)
```

entire dictionary will deleted

* dictionary Copy :-

```
d={a: "apple", b= "ball",
   c= "cat"}
print(d)
d1 = d
print (d1)
```

→ print key values

```
print (d.keys())
```

→ print values

```
print(d.values())
```

→ print items

```
print(d.items())
```

* membership tests

→ It is applicable only for 'keys' not for values

```
print (a in d) = True, key
print (cat in d) = False, Value
```

Creating dict by using lists:-

```
keys = [1, 2, 3, 4]
values = ["mohan", "sai", "durga", "raju"]
d = dict(zip(keys, values))
print(d)
```

o/p: {1: "mohan", 2: "sai", 3: "durga", 4: "raju"}

→ W.A.P a number in range (or) not

→ W.A.P to find sum of all numbers in list.

→ WAP takes i/p as list and returns new list with unique elements of the first list.

```
list = []
total = sum(list)
```

functions in python:

i) predefined (or) builtin functions:

print, id, type, input, int, len.

ii) user defined functions:

Syntax: def function_name(parameters):
    docstring
    statements
    return statement

(def = keyword)

function name = identifier.
parameters = i/p values, optional
docstring = descriptions

Ex:-
```
def f1(name):
    print("hello", name)
f1("mohan")
```
o/p:- hello mohan.

```
f1("durga")
```
→ hello durga

* squaring of numbers:

```
def square(n):
    Print("square is", n**n)
square(5)
```
o/p: 25

* squaring in one-line:

```
n = int(input("Enter a number"))
def square(n):
    print("square is", n**n)
square(n)
```
o/p: Enter number = 7
square is = 49

* adding two numbers:

```
def sum(x, y):
    Print(x+y)
sum(7, 8)
```
o/p: 15

Even (or) odd:

```
def even_odd (n):
    if n%2==0:
        print (n,"is even")
    else:
        print (n,"is odd")
even_odd(4)
```

o/p: 4 is even

function without parameters:

```
def f1():
    print("hello")
f1()
```

o/p: hello

W.A.P to print Arithmetic operations by creating four functions

(23)

```
def add(a,b):
    return (a+b)
def sub(a,b):
    return a-b
def mul(a,b):
    return a*b
def div(a,b):
    return a/b
print("select choice")
print("1: add")
print("2. sub")
print("3. mul")
print("4. div")
choice = int(input("Enter your choice"))
if choice >0 and choice <=4:
    a= int(input("Enter num1"))
    b= int(input("Enter num2"))
    if choice==1:
        print("sum is", add(a,b))
```

elif choice == 2 :
    Print ("sub is", sub(a,b))
elif choice == 3 :
    Print ("mul is", mul(a,b))
elif choice == 4 :
    Print ("div is", div(a,b))
else :
    Print ("Invalid choice")

(22)

** Types of arguments :

→ Let us Consider the functions

def add (a,b) :
    Print (a+b)
add (20,10)

→ Here (20,10) are actual arguments
(a,b) are formal arguments

→ Four types of actual arguments allowed in Python :

i) positional arguments
ii) keyword "
iii) default arguments
iv) Variable length arguments

(i) positional arguments :-

The arguments which are passed to a function in Correct positional order is called positional arguments

→ If we change order (or) position of arguments, result may be change

→ If increase no. of arguments it will give type error

Ex: def sub (a,b) :
    Print (a-b)
sub (20,10) = 10
sub (10,20) = -10
sub (10,20,30) = Type error

ii) keyword arguments : (30)

→ The arguments which are passed to a function by using keyword (or) parameter name is called keyword arguments.

→ while working with F.A The order (or) position are not important, keywords are important.

→ We can also use one positional argument and one keyword argument but make sure that first we have to mention positional arguments after F.A, otherwise will get syntax error.

Ex: def fi(name, msg):
    print("hello", name, msg)

fi(name = "mohan", msg = "good evng")

fi(msg = "good evng", name = "mohan")

positional arguments :

fi("mohan", "good evng")

fi("good evng", "mohan")

one positional, one keyword :

fi("mohan", msg = "good evng")

fi("good evng", name = "mohan")

→ Syntax error

iii) default arguments :-

→ In general, when we have a function with argument, while calling that function we should pass i/p value, otherwise will get error.

→ when function is having default argument, we need not to pass any value to function, function will work by taking default value.

→ default argument is only Considered when we not supply value to the functions.

→ Though the function is having default argument, if you supply the value, supplied value only will been Considered.

Ex:-

```
def fi(Course = "python"):
    print("my Course is", Course)

fi("c") = c
fi("net") = net
fi() = python
fi("java") = Java
fi() => python
```

→ Note: we can also pass one default argument and non-default argument

→ but always non-default argument should be first then after default argument otherwise it will give error.

Ex:

```
def f1(name, Course="python"):
    Print(name, "Course is", Course)
```

```
f1("mohan", "c")
f1("raj", "net")
f1("hari")
f1("manoj", "java")
f1("durga")
```

o/p:
mohan Course is c
raj Course is net
hari Course is python
manoj Course is java
durga Course is python

iv) Variable-length arguments :

Some time we can pass a no. of Variable arguments to function.

→ Variable-length argument can been declared like

```
def f1(*n):
    print(n)
```

→ Variable length arguments will accept any no of values including "0" number

Ex:

```
def f1(*n):
    Print(n)
f1()
f1(10)
f1(10, 20)
f1(10, 20, 30)
f1(10, 20, 30 ---- n).
```

what is *args and **kwargs?

→ *args allows to pass n'of argument to functions

→ *args is just name Convention, we can take any valid name instead of args, like, *n, *a, *anas ---

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

The above function will accept only 2 values, what happen if we pass more than 2 values, it raise error, to avoid that we use *args.

Ex: def add(*args):

```
    s=0
    for i in args:
        s = s+i
    print("Sum is", s)
add(10, 20, 30)
add(10, 20, 30, 40, 50 --- n)
```

→ **kwargs allows to pass no. of keyword arguments to a function.

Eg: def fn(a,b,c,d):
    print(a,b,c,d)
kwargs = {'a': "apple", 'b': "ball", 'c': "cat", 'd': "dog"}
fn(**kwargs)

o/p: apple ball cat dog

* W.A.P to create and print list where the values b/w (1 & 20) are square

* W.A.P that accepts hyphen seperated sequence of words as i/p and prints that words in a hyphen seperated sequence after sorting them alphabetically.

Bytes datatype:

→ It represent a group of byte numbers just like an array

x = [10, 20, 30, 40]
b = bytes(x)
for i in b:
    print(i)

o/p:
10
20
30
40

* bytes must be in the range of (0 to 255) both 0 & 255 included

→ byte is immutable

→ byte must be in range of (0 to 255)

byte array:

→ Except that its values can been modified

x = [10, 20, 30, 255]

b = bytearray(x)

b[0] = 100

for i in b:

    print(i)

o/p 10
20
30
255

→ byte array is mutable it can been modified

→ It is also range in (0 to 255)

Frozenset:

s = {10, 20, 30, 40}

fs = frozenset(s)

print(fs).

→ It is unordered

→ It cannot modified

Command-line arguments:

→ which passed at the time of Execution.

→ To work this we use a Variable called "argv"

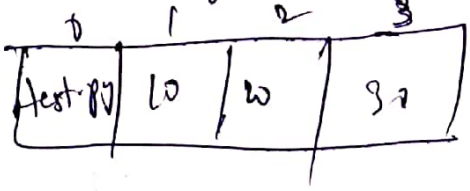
→ here 'argv' is not an array it is list

→ 'argv' is present in 'sys' module.

Ex: G:\pythonoline@spin — py (test.py 10 20 30)

↓

There are Command-line arguments



argv[0] = test.py
argv[1] = 10
argv[2] = 20
argv[3] = 30

cd pythonwalkthrough @ 3pm
py test.py

Ent

Indexes
a = int(argv[1])
b = int(argv[2])
Print(a+b)

Values through command line:

Sum = 0
args = argv[1:]
for x in args:
    n = int(x)
    Sum = Sum+n
Print("Sum is", Sum)

O/P: 10, 20, 30, 40 → n

100

Types of Variables in functions:

i) local Variable
ii) Global Variable.

(36)

i) local Variables:

The Variables which are declared inside the function and that Variables are available to only that function is called local Variable

→ local Variable can't be accessed outside of the function

ii) Global Variables

The Variables which are declared outside of the function and that are available to all that function in that program is called global Variable

Eg: for local Variable

```
def f1():
    a = 10
    print(a)
def f2():
    b = 20
    print(b)
f1()
f2()
```

O/P: 10
20

Ex: for global Variables:

```
a=10
def f1():
    print(a)
def f2():
    Print(a)

f1()
f2()
```

o/p:- 10
10

* global keyword:-

global keyword can be used to make a global variable available to the function for required modification.

Ex:

```
a=10
def f1():
    global a
    a=99
    Print(a)
def f2():
    Print(a)
f1()
f2()
```

o/p:- 99
99

→ If local Variable name and global Variable name is same, then to access global Variable value inside a particular function we use "globals" function

Ex:

```
a=10
def f1():
    a=99
    Print(a)
    Print(globals()['a'])
def f2()
    Print(a)
```

o/p:- 99
10
10

f1()
f2()

* function aliasing:-

→ Giving other name to a function is called function aliasing

→ If we delete the name of the function, we have other name to access.

```
def f1():
    print("hello")
f2 = f1
del f1
f1()
f2()
```

o/p:- hello, because f1() is deleted

```
f3 = f2
del f2
f2()
f3()
```

o/p:- hello, because f2 is deleted

Anonymous (or) lambda functions:

→ Sometimes we can declare a function without any name, such kind of nameless functions are called anonymous (or) lambda functions

→ Using lambda we can write very Concise Code.
(small)

→ Using lambda we can reduce no. of lines of code and improve readability of the program

→ lambda are instantly used. only one time usage

→ Some functions will except other functions as an argument (or) some times we can pass a function as an argument to other functions in this case lambda's are best choice.

Syntax:

lambda argument list: Expression

→ lambda is a keyword

→ by default lambda will returns Expression

value internally so, that we don't use return keyword.

Ex: s = lambda n: n*n
print(s(5))
o/p: 5*5 = 25

To find biggest of two numbers using lambda

s = lambda a,b : a if a>b else b
print(s(89, 67))
o/p: 89.

To find sum of two numbers

s = lambda a,b: a+b
print(s(45+40))
o/p: 85

To find even (or) odd

s = lambda n: "is even" if n%2==0 else "is odd"
print(s(6)) o/p: even
print(s(9)) odd.

1. filter()
2. map()
3. reduce()

→ The above functions will except other functions as arguments

(i) filter():

This function is used to filter the values from given sequence based on Condition.

Syntax:

"filter(function, sequence)"

→ function is for Conditional check

→ sequence, list (or) tuple (or) set. is the

* To filter the even numbers from given sequence without lambda *

```
def iseven(x):
    if x%2==0:
        return True
    else:
        return False
l1=[2,3,4,5,6,7,8,9]
l2=list(filter(iseven,l1))
print(l2)
```

o/p: [2,4,6,8]

→ with lambda function:-

```
l1=[2,3,4,5,6,7,8,9]
l2=list(filter(lambda x:x%2==0,l1))
print(l2)
```

o/p: [2,4,6,8]

(ii) map():-

For every element present in sequence apply some condition and return new sequence of elements for this purpose we use "map()" function.

Syntax:

map(function, sequence).

* To double the list elements using map without lambda *

```
def dbl(x):
    return 2*x
l1=[2,3,4,5,6,7,8,9]
l2=list(map(dbl,l1))
print(l2)
```

o/p: [4,6,8,10,12,14,16,18]

→ with lambda function:

(41)

```
l1 = [2, 3, 4, 5, 6, 7, 8, 9]
l2 = list(map(lambda x: 2*x, l1))
print(l2)
```

o/p: [4, 6, 8, 10, 12, 14, 16, 18].

→ Note:- We can also use two lists on map function but, make sure that two lists should have same number.

Ex:

```
l1 = [2, 3, 4, 5]
l2 = [5, 6, 7, 8]
l3 = list(map(lambda x, y: x*y, l1, l2))
print(l3)
```

o/p: [10, 18, 28, 40].

* Reduce():

To reduce the elements from sequence and returns one value

→ reduce function is present in ["functools"] module

Ex:

```
from functools import *
l1 = [2, 3, 4, 5, 6, 7, 8, 9]
l2 = reduce(lambda x, y: x+y, l1)
print(l2)
```

o/p: 44

* Nested functions:

→ A function which is having one (or) more functions with in it

Ex:-1:

```
def f1():
    Print("hello")
    def f2():
        Print("hai")
    f2()
f1()
```

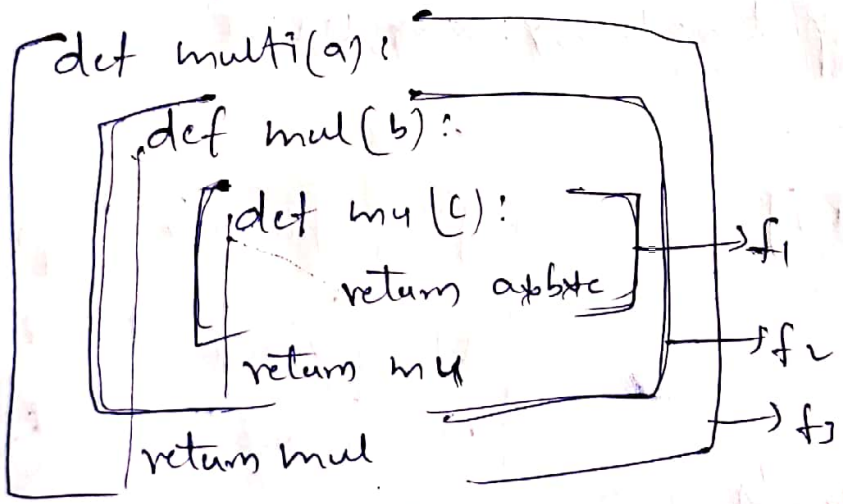
o/p: hello
hai

factorial:

factorial(n) = n * factorial(n-1)
(3) = 3 * factorial(2)
= 3 * 2 * factorial(1)
= 3 * 2 * 1 * factorial(0)
= 3 * 2 * 1 * (1)
= 6

Ex:2:

```
def multi(a):
    def mul(b):
        def m4(c):
            return a*b*c
        return m4
    return mul
m = multi(2)(3)(4)
Print(m)
```



o/p: 24.

Recursive function:

A function that call's itself is called recursive function

Ex: factorial(n) = n * factorial(n-1)

(3) = 3 * factorial(2)

= 3 * 2 * factorial(1)

= 3 * 2 * 1 * factorial(0)

= 3 * 2 * 1 * 1

= 6

Ex: Program:

```
def factorial(n):
    if n==0:
        result=1
    else:
        result = n * factorial(n-1)
    return result
print("factorial of 3 is :",factorial(3))
```

o/p: factorial of 3 is 6

Modules in python

→ A group of variables, functions, classes can store into a file is called module

→ In python every "py" file act as a module

→ The main importance of modules is reusability.

→ modules can be of two types

i) user defined modules

ii) pre defined (or) built-in modules

→ The modules which are created by programmer and that can been used according to their requirement is called user defined modules

→ built-in modules will come automatically when install python software

test.py:

```
x = 100
def add(a,b):
    print("sum is", a+b)
def sub(a,b):
    print("sub is", a-b)
```

test1.py:

(111)

```
import test
print(test.x)
test.add(20,30)
test.sub(45,40)
```

o/p: 50
sum is 50
sub is 5

→ In "test.py" x, add, sub are members of the module

→ To access members of the module we use (modulename.membername)

module aliasing:-

Giving other name to a module is called module aliasing

→ Here we give the alias name of the module we can only use alias name to access members.

* test2.py:

import test as t
Print (t.x)
t.add (3,5)
t.sub (5,1)

o/p: 8
4

* from ---- import :

→ The main advantage of "from -- import" is we need not use any module name (or) alaise name to access members

* test3.py:-

from test import *
from test import x, add, sub
Print (x)
add (8,9)
sub (3,1).

* Importing members from two different modules :-

→ test4.py: (145)

empid = 123
def display ():
    Print ("hello")

→ test5.py:

x = 100
def add (a,b):
    Print ("sum is", a+b).

→ test6.py:

import test4, test5
Print (test4.x)
Print (test4.empid)
test4.add (2,3)
test4.sub (3,1)
test5.display ().

* Assume that two modules are having same variables

→ test7.py:

a = 100

→ test8.py:

a = 200

→ test9.py:

from test7 import *

from test8 import *

print(a)

print(a)

→ when we execute "test9.py" the result is 'a' values 200, 200 both times, that means 'a' value only considers from "test8.py".

→ To avoid this conflict

from test7 import *

print(a)

from test8 import *

print(a) o/p: 100, 200

And other version is:

from test7 import *

print(a)

from test8 import a as b

print(b)

o/p: 100, 200

(116)

* import test7, test8

print(test7.a, test8.a)

(or)

print(test7.a)

print(test8.a)

o/p: 100, 200

100

200

* different ways to import modules *

1. import modulename
2. import modulename as m1
3. from ~~import~~ modulename import *
4. from modulename import member1, member2

5. from modulename import
membr1 as m1, member2
as m2

6 import module1, module2.

→ To get particular module description (or) information

#import test
help(test) #

→ To get all modules list

#help("modules") #

→ python supported programs

items = [n for n in input("Enter items seperated by").split('-')]
item.sort()
print '-'.join(items)

o/p:- i/p: mohan-durga-soft-sai

o/p: durga-mohan-sai-soft.

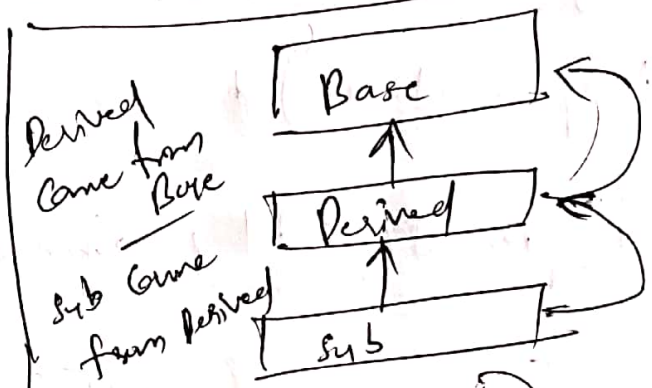
* f1(name = 'mohan', msg = 'good evng') → keyword argument (name, msg) are key

* f1('mohan', 'good evng') → positional argument because their is no keys.

* Encapsulation :-

It Cannot access the private Variable and Cannot modify the private Variable Outside the class it is called as the encapsulation, so it Can modify Through public method.

(--- Anna.) private class



* Inheritance :-

Class Baseclass():

class Derivedclass(Baseclass):

} Single Inheritance

class Subclass(Derivedclass):

} → Multi level Inherita

s = Subclass

So ——

multiple inheritance :-

```
class A:
    ==
class B:
    ==
class C(A,B):
    ==
```

} multiple inheritance

i) operator overloading :

magic methods are used

```
__add__(self, other):
```

ii) method overloading :

```
def m1(self)
    ==
def m1(self,a)
    ==
def m1(self, a, b)
    ==
```

} not possible because same method name

iii) Constructor overloading :

```
def __init__(self):
    ==
def __init__(self,a):
    ==
def __init__ self(a,b)
```

```
print(list(range(2,34)))
```

= [2,3,4,5 ——— 33].

→ lst.sort()

```
print(lst)
```

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

= Ascending order

→ lst.sort(reverse = True)

```
print(lst)
```

= [5,4,3,2,1] = decending order

→ lst2 = lst1

```
print(lst2)
```

= copying of list1 to list2

→ creating list by user

```
lst []
item1 = int(input("Enter int item"))
item2 = input("Enter string"))
item3 = float(input("Enter float"))
lst.extend([item1, item2, item3])
print(lst).
```

= [34, sai, 45.67].

The shortest way to SUCCESS is HARDWORK

SOFTQ SOLUTIONS
We make IT easy

SOFTQ

COMPUTER EDUCATION

we make IT easy...