Strings

- + operator for concatenation
- * operator for repetition.
- Replacing a string with another string
  s.replace(old string, new string)
- Joining of strings:
  s = seperator.join(group of strings)

List

- Split function divides the string into list
- append() → to add item at the end
- To insert item at a specified position:-
  n.insert(1, 888).
  (Inserting 888 at the 1th index.
- Extend:- To add all items of one list to another list. l1.extend(l2)
- Remove() function:- To remove specified item from the list. If item present multiple times then only first occurence removed.
  n.remove(10).
- Pop() function:- It removes and returns the last element of the list.
- n.pop() - To remove and return last element of the list.
- n.pop(index) - To remove at specified index.

List Comprehension:

List = [expression for item in list if condition]

Tuple:

- Tuple Comprehension is not supported in python.
- Paranthesis are optional for tuple.
  t = (10,20,30,40)
  t = 10,20,30,40
- adding value to a tuple a.
  a + (4,)

Sets:-

- add(x) → adds item to a set.
- update(x,y,z) → to add multiple items to the set where x,y,z are not individual items but iterable objects like lists, range etc.
- pop():- removes and returns some random element from the set.
- remove:- It removes specified elements from the set
- discard:- same like remove but we wont get error if element is not present in the set.
- clear():- To remove all the elements from the set.

- Union() :- To return all elements present in both the sets.
- intersection() :- Returns common elements present in both x and y.
- difference() :- Returns the elements (x-y) present in x and not present in y.
- Symmetric_difference() or x^y :- Returns elements present in either x or y but not in both. (uncommon elements)
- Set comprehension is possible

- Dictionary :-
- get() :- To get the value associated with the key.
- pop() :- It removes the entry associated with the specified key and returns the corresponding values. d.pop(key)
- popitem() :- It removes an arbitrary item(key-value) from the dictionary and returns it. d.popitem()
- .keys(), .values(), .items()
- Set default() :-

If the key is already available then this function returns the corresponding value. If the key is not available then the specified

d.setdefault(k, v)

- d.update(x)

All items present in the dictionary x will be added to dictionary d.

- Dictionary Comprehensions are available.

Property	List	tuple	set	Dictionary	Strings
Insertion Order	✓	✓	✗	✗	✓ (mixture of values)
Duplicate Values	✓	✓	✗	✗	✓
Heterogeneous Values	✓	✓	✓	✓	✓
Immutable	Mutable	Immutable	Mutable	Mutable	Immutable
Indexing and Slicing	✓	✓	✗	✗	✓

NPV Important Syntaxes :-

- np.array(list) :- To create a numpy array from any list
- array.shape gives the number of rows and columns.
- array.size gives the number of elements present inside.
- list(range(10)) gives numbers from 0 to 9.
- array.itemsize prints memory occupied by each element.
- array.dtype gives us the data type.
- np.eye(m,n) is used when rows ≠ columns in matrix.
- array.round(4) means 4 places after decimal point. (rounding)
- np.arange(21) gives values from 0 to 20.
- @ (or) .dot() used to perform matrix multiplication.
- sum() returns the addition of all the elements.
- power() function raises the value. (2 parameters taken)
- pd.show_versions() shows the version of the library.
- with index, we can set the indexes.
- with columns we can set the columns.
- rename function can be used to rename something.
- .iloc is position based indexing.
- .loc is name based indexing.
- np.linspace() gives the equal number of intervals.
- seed() helps us to standardize the values.
- np.remainder(x,8) gives the remainder when divide by 8.

- np.identity() gives the identity matrix.
- np.array(M).T is used to transpose the matrix.
- Random Module in Python
- r.random() → generates a random float number between 0.0 to 1.0.
- r.randint() → generates a random integer between the specified integers. It takes two arguments.

For example:- r.randint(5,10), the output can be 5(or) 6(or) 7(or) 8(or) 9(or) 10

- r.randrange() → Returns a randomly selected element from the range created by the start, stop and step arguments.
- r.choice() → Returns a randomly selected element from a non empty sequence.
- r.shuffle() → This function randomly reorders the elements in list.

Numpy random module:-

- np.random.random(size parameter) returns random num from uniform dist over the half open interval [0.0, 1.0)
- np.random.rand() → we can give any required shape inside. It creates an array of the given shape and populates it with random variables derived from a uniform distribution b/w (0,1)
- np.random.rand(3,3,2) means generate 3 arrays with the shape of each array as (3,2).
- np.random.randn() returns the values from the standard normal distribution with mean 0 and standard deviation 1.

- np.random.randint(5,20) # Returns one random number between the values 5 and 19 (20 is excluded).
- np.random.randint(20,50,5) # Returns 5 random integers b/w 20 and 49 (50 is excluded).

- np.arange(10, 20, 2)
  output :- array([10, 12, 14, 16, 18])
- np.linspace(0,50,20) gives 20 intervals starting from 0 and ending with 50 (included).

- import keyword
  keyword.kwlist → 33 reserved keywords.
- eval() means evaluate
  x = eval("10+20+30")
  print(x)
  → 60
- argv is available in sys module.
- Python has 7 types of operators :-

→ Arithmetic Operators
→ Relational operators.
→ Assignment operators.
→ Logical Operators.
→ Membership Operators
→ Identity Operators
→ Bitwise Operators

- print() → to print
- type() → to know the data types
- In python everything is an object.
- a=10 means a is the reference variable pointing to the object 10
- id(a) means to find address of object a.

- Decimal form (Base 10)
  Binary form
  Octal form
  Hexa decimal form

} ways we can represent int datatype

- Decimal (Base 10)
  allowed values :- 0 to 9.
- Binary (Base 2)
  allowed values :- 0 and 1

  | If any value starts with 0b (or) 0B, then it is considered as a binary number. (zero b (or) zero B)

- Octal (Base 8)
  allowed values :- 0 to 7.

  | 0 followed by small or capital O
  a = 00777

- Hexa decimal (Base 16)
  allowed values :- 0 to 9
  a to f (or) A to F

  | 0x or 0X
  (zero with small x or X).
  a = 0XFace → 64206
  a = 0XBeef → 48879

In jupyter,

a = 0o0101
a
output :- 65

a = 0b0011
a
output :- 3

oct(65) = '0o101'

bin(3) = '0b11'

Session 8 :-

Bytes type :-

Represents a group of byte numbers.

```
x = [10,20,30,40]
b = bytes(x)
type(b)
→ byte
```

```
b[0] → 10
b[1] → 20
b[-1] → 40
```

. In the bytes data type every value should be in the range 0 to 256 only.

```
x = [10,20,256,258]
→ Error
```

. Once the list is converted into bytes, then it is immutable.

```
. b[0] = 120
→ error
```

(bytes doesn't support item assignment)

. Bytes data type is immutable.

Byte array :-

Bytes and bytesarray are same.

But, Byte array is mutable.

'In python None can be added (appended) into the list.

Session 9 :-

. range datatype represents a sequence of values.

range(end)

→ values from 0 to (end-1).

Range is immutable

```
r = range(10)
type(r)
r
→ range(0,10)
for i in r: print(i)
→ 0
  1
  2
  3
  4
  5
  6
  7
  8
  9
```

(slicing and indexing are for ordered sequence)

frozenset :-

```
s = {10, 20, 30, 40}
fs = frozenset(s)
type(fs)
→ <class 'frozenset'>
fs
frozenset({40, 10, 20, 30})
fs.add(50)
→ (error)  we cannot make changes in the frozenset.
```

· {} is treated as an empty dictionary.

· set() is treated as an empty set.

· duplicate keys not allowed in dictionary.

Session 10

· Range data type is always immutable.

```
· def f1():
    pass
→ (error)
```

using pass function

```
· def f1():
    pass
→ (not an error)
```

· another example :-

```
if amount > 10000:
    do some action
else:
    pass
```

· Escape characters \n, \t

Example:

```
s = "durga\n software"
⇒ durga
  software
```

```
s = "durga\t software"
⇒ durga  software
```

Bitwise Operators

We can apply these operators bitwise

& (AND)

| (OR)

^ (XOR)

~ (Compliment operator)

<< (left shift operator)

>> (right shift operator)

We can apply these operators for int type and boolean type.

& ==> If both bits are 1, then only 1 otherwise 0

| ==> If atleast one bit is 1 then 1 otherwise 0.

^ ==> If both are different then 1 otherwise 0.

~ ==> bitwise compliment operator

1 ==> 0 and 0 ==> 1

(One is replaced with 0 and 0 is replaced with 1)

- 4&5 ==> valid
- True & True ==> Valid
- 10.5 & 2.6 ==> Invalid

- 4&5 ==> 4

Apply binary representation for 4 and 5,

4 → 100

5 → 101

100 → Answer is 4

- 4|5 ⇒ 5

4 → 100

5 → 101

101 → Answer is 5

- 4^5 ⇒ 1

4 → 100

5 → 101

001 → 1

- Bitwise Complement operator (~):-

~4 ⇒ -5

Background Calculations:-

- Assume 4 is going to be represented

000000000.....100 → ①

↓

0 means +ve number and 1 means negative number

- Positive numbers will be reflected directly into the memory, but negative numbers will be represented in 2's complement form.

Now lets apply negation of 4 (~4) to ①,

~4 ⇒ 111111111....011

↓

most significant digit is 1 (-ve) value.

Remaining values will be represented in 2's complement form.

2's complement = 1's complement + 1

1's complement means 0 will be replaced with 1 and 1 is represented with 0.

Class 14

Special operators:-

1) Identity operators (is and is not)
2) Membership operators (in and not in)

Test.py → ⓐ
To execute in command prompt.

id is a function (inbuilt)

List 1 = [10, 20, 30]
List 2 = [10, 20, 30]
print (id (list 1))
print (id (list 2))
} Both the address are different. Physically. The object may have the same content. But there are two different reference variables. ⓑ

print (list 1 is list 2)
(reference comparison)
→ False
print (list 1 == list 2)
→ True
} The '==' is comparing the content inside the object.
(Content Comparison)

Space is a character in python

- Priority order:-
  - unary operators ~x
  - binary operators x+y
  - ternary 3 arguments

General priority order irrespective of language.
- Paranthesis operators
- Unary operators
- binary operators
- ternary operators
- Assignment operators

- Operator Precedence in Python:-
  - () → Paranthesis
  - ** → exponential
  - ~, - → unary operator
  - *, /, %, //
  - +, -
  - <<, >>
  - &
  - ^
  - |
  - >, >=, <, <=, ==, !=
  - Assignment operators
  - is, is not
  - in, not in
  - not
  - and
  - or

a = 30
b = 20
c = 10
d = 5

print ((a+b) * c / d)
→ 100.0

In python division operator is always used to generate float value only.

module → a group of functions.
Library → a group of modules.

Input and Output Statements :- | Session 15 |

Because we require to read data from the keyboard and we require to print data to the console.

This story is for Python 2

- Read data from the keyboard → Dynamic data :

Two functions are available to read dynamic data.

→ raw_input ("Enter some number") → always is considered as str (required type casting)

input ("Enter some number") → not considered as str type. (not required type casting)

- [int(x) for x in input ("Enter 2 numbers :").split()] (Space seperation)

a,b =

→ we get the list of two int values.

The 'enter two numbers' part is considered as one string and split divides it.

- Read 2 float values from the keyboard which are specified with , seperation and print sum.

input("Enter 2 float values :")

Solution :-

```
a,b = [float(x) for x in input("Enter 2 float values :").split(',')]
print("The sum :", a+b)
```

- eval() : means evaluate

```
x = eval("10+20+30")
print(x)
→ 60
```

Eval is going to convert into internal data type automatically.

Command Line Arguments (The arguments which are passing from the command prompt)

- py test.py 10,20,30,40,50 → These values are considered as Command line arguments.

CLA

- argv (internally this value holds all the command line values).
- argv is an array (one option) ✗

argv maybe a list (second option) ✓

↓

argv ⇒ list type. (argv is a list type)

- from sys import argv
  print(type(argv))
  → list

argv is available in sys module

- The arguments which are passing from command prompt, these arguments by default considered as command line arguments.

- Example:- In command prompt

py	test.py	10	20	30

There are four values

```
from sys import argv
print(argv)
→['test.py','10','20','30']
```

Section 16

- len function :- To find the number of elements.
- # read a group of int values from the keyboard as cmd line arguments and print sum.

```
→ from sys import argv
  args = argv[1:]
  sum = 0
  for x in args:
      n = int(x)
      sum = sum + n
  print("The sum:", sum)
```

(usually space is the seperator in CLA)

(enclose with double quotes if CLA content contains space)

(By default CLA datatype is string)

Class 32

Functions :-

1) Built in Functions (pre-defined functions)
2) User Defined Functions

The benefits of functions is code reusability.

Class List
def append():

- pop() is a method but not a function.
- append() is available in class list.
- If any function defined inside a class, then they are called methods.

```
def calc(a,b):
    sum = a+b
    sub = a-b
    mul = a*b
    div = a/b
    return sum, sub,
           mul, div
```

- Parameters :- (Types of arguments)
  - positional parameters
  - keyword parameters
  - default parameters
  - Var arg parameters (variable length arguments).
- We can pass parameters to the functions.
- ~~positional arguments follows keyword arguments.~~
- We can take default arguments at last (or) after the non-default arguments.
- After keyword arguments, we should not take positional arguments.

- Variable length arguments :- (Var arg methods)
- WAP to print sum of given numbers.

```
def sum(*n)
    result = 0
    for x in n:
        result = result + x
    print("The Sum:", result)
```

n is stored as tuple.

- After variable length arguments, if we are taking any other arguments, we should provide as keyword arguments parameter.

Example :-

```
def insertemployee(name, age, empno, esal, eaddr):
    insert into employees table
    insert into payments table
    insert into projects table

for e in employees:
    insertemployee(data)
```

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

→ hello
None

Parameters in Python :-

- positional arguments
- key word arguments
- default arguments
- var arg (Variable length arguments)

from this example a, b are positional arguments.

- def calc(a,b) → positional arguments

calc(a=100, b=50)
calc(b=50, a=100) } key word arguments

Example :-

```
def calc(a,b):
    calc(100,b=50) ✓
    calc(b=50, 100) ✗ → we shouldnt take positional argument after keyword argument.
    calc(50,a=100) ✓ ✗ → ~~we should take default arguments at last.~~
                         The error is the value of a is repeated.
```

Example :-

```
def wish(name = "Guest", msg) : pass
```

This is error because we should take default arguments at last. (or) after the non-default argument.

This is Valid :- `def wish(msg, name = "Guest") : pass`

Variable length arguments :-

Example :-

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

sum(10,20) Till here alright.

.... afterwords, our requirement is,

sum(10,20,30)

sum(10,20,30,40)

we cannot use this, before previously created function has two arguments.

For increase in the arguments, we must always create new functions.

To solve this issue, we must go for Vararg.

We can declare variable length argument in this way.

```
def sum(*n)
```

Here * means any number of arguments.
n means group of values

Now we can pass any number of arguments.

Here n is a tuple internally.

Example :- → var arg argument

```
def sum(*n):
    result = 0
    for x in n:
        result = result + x
    print("The sum :", result)
```

. Now we can use the above function to solve sum(10,20,30), sum(10,20) etc.

Example :-

```
def sum(name, *n):
```

This will work.

(above example)

Example :-

```
def sum(*n, name)
```

This is error.

But, if we take the key word argument,

sum(10,20, name = "Vishal") → it will work.

Keyword variable length arguments:

Different types of keyword arguments with different names. If we want to display such different kind of keyword arg we go for this. (**kwargs)

Internally **kwargs is dictionary.

Example:

```
def display (**kwargs):
    print ("Record Information:")
    for k,v in kwargs.items():
        print (k,"....",v)

display (name="Durga", marks=100, age=48, GF="Sunny")
```

Example:

```
def f(arg1, arg2, arg3=4, arg4=8):
    print (arg1, arg2, arg3, arg4)
```

In this example arg3=4 and arg4=8 are default arguments.

f(3,4)
It returns 3, 4, 4, 8.

f(10,20,30,40)
It returns 10,20,30,40

f(25,50, arg4=100)
It returns 25,50,4,100

f(arg4=2, arg1=3, arg2=4)
It returns 3,4,4,2

f() is invalid.

f(arg3=10, arg4=40, 20, 30)
gives error because after keyword argument, positional argument is taken

f(4,5, arg2=6)
It is error, because arg2 is repeating.

f(4,5, arg3=5, arg5=6)
This is error because of unexpected keyword argument arg5.

- Local variable is declared inside a function whereas Global variable is declared outside a function.

```
a=10          -> Global variable
def something():  -> Local variable
    a=15
    print(a)
print(a)
```

Class 33 :-

- Functions is a group of statements
- Module is a group of functions.
- Package is a group of modules.
- A group of packages is library.

Recursive Function :- A function that calls itself is called recursive function.

Advantages : We can reduce length of the code and improves readability.
We can solve complex ~~functions~~ problems.

Example :-
Write a function to find the factorial using recursion.

```
def factorial(n):
    if n==0:
        result = 1
    else:
        result = n * factorial(n-1)
    return result
```

Anonymous Functions :- nameless functions are known as anonymous functions.
Used for instant use (only one time usage).

Let us consider a normal function :-

~~def factorial~~ def squareit(n):
    return n*n

But we want to write a concise code, we use lambda function.

lambda n : n * n
(n is the input argument we are taking.)
(we must return this value.)

The meaning of the above statement is, if we provide n, then it is going to return n*n.

```
s = lambda n : n*n
print (s(4))
```

output is 16.

lambda is a keyword to return anonymous function.
In lambda, there is implicit return statement.

lambda input : expression

- WAP to write lambda for square of given number.

```
s = lambda x : x*x
```

- To find the sum of two given numbers.

```
s = lambda a,b : a+b
print("sum of {} and {} is : {}".format(2,4, s(2,4)))
```

- To find the biggest of two values.

```
s = lambda a,b : a if a>b else b;
```

The above expression is ternary operator.

```
print("the biggest of {} and {} is : {}".format(10,20,s(10,20)))
```

- we can pass a function as an argument to another function. we can use lambda function.

---

filter()
map()
reduce()
} These functions always expecting another function as argument.

- filter() :-

Syntax :-

filter (function, sequence)

This value is always considered true (or) false.
If false, the sequence is not executed

For every element in the sequence, function is applied.

```
def iseven(x):
    if x%2==0:
        return True
    else
        return False

l=[0,5,10,15,20,25,30]
filter(iseven, l)
l1 = list(filter(iseven, l))
print(l1)
```

Type of l1 is filter.

We use the lambda function for the above example.

(next page)

- l = [0,5,10,15,20,25,30]
  l1 = list (filter (lambda x : x%.2 == 0, l))
  print (l1)

- If we need odd numbers then,
  l2 = list(filter (lambda x : x%.2 != 0, l))
  print (l2)

- To filter a group of values based on some condition, we can go for filter method.

- map() method :-
  For every value, to provide some equivalent value by applying some function, we use map() method.

  map (function, sequence)

- For every element present in the given sequence, apply some functionality and generate new element with the required modification.

Example :-

```
def double (x):
    return 2*x
l = [1,2,3,4,5]
l1 = list (map (double, l))
print (l1)
```

Instead, we use lambda function, we get :-

```
l = [1,2,3,4,5]
l1 = list (map (lambda x : 2*x, l))
print (l1)
```

- We can apply map function on multiple sequences also. Both the sequence must contain same number of elements.

```
l1 = [1,2,3,4]
l2 = [10,20,30,40]
l3 = list(map (lambda x,y : x*y, l1, l2))
print (l3)
```

Nested Functions :- The funtions declared inside another funtion.

```
def f1():
    def inner(a,b):
        print("The Sum:", a+b)
        print("The average:", (a+b)/2)
    print()
    inner(10,20)
    inner(20,30)
f1()
```

inner is defined inside f1() and also called inside f1.

Inner is repeatedly required in f1(). That is when nested functions is useful.

- A funtion can return another funtion.

Example :-

```
def outer():
    print("outer funtion started")
    def inner():
        print("inner funtion execution")
    print("outer funtion returning inner funtion")
    return inner;

f1 = outer()
f1()
f1()
```

(If we use inner(), we get none output)

This is funtion call, the returned value is assigned to f1.

- Meaning of f1 = outer() is we are calling outer function. Outer funtion will return inner funtion. That inner funtion we are going to assign with f1.
- Meaning of f1 = outer, for outer funtion we are giving another name.
  → funtion aliasing.

---

reduce()

reduce() funtion reduces sequence of elements into a single element by applying the specified funtion.

reduce(funtion, sequence)

reduce() funtion is present in funtool module and we should import it.

Shallow Copy and Deep Copy :-

list1 = [1,2,3,4]

list2 = list1

list 2

output is [1,2,34]

Now, we change a value in list2.

~~list[1] = 1000~~ list 2[1] = 1000

list2

output is [1, 1000, 3, 4]

list1

output is [1, 1000, 3, 4]

=, copy(), deepCopy()

Here, this is = (equals to) Copy.

when we make changes in list 2, they get reflected in list1.

Because,

Both list1 and list2 are pointing to the same memory location.

Shallow Copy copy()

- list1 = [1,2,3,4]

list2 = list1.copy()

This is because, both have different memory locations.

list2

output is [1,2,3,4]

Changing values in list2.

list2[1] = 1000

list2, list1

Output is ([1,1000,3,4], [1,2,3,4])

[redacted] Shallow Copy [redacted]. Shallow copy of nested list :-

- list1 = [[1,2,3,4], [5,6,7,8]]

list2 = list1.copy.

---

list1[1][0] = 1000

Now, changes get reflected in both list1 and list2,

that is, [[1,2,34], [1000, 6, 7, 8]]

This is known as [redacted]. Shallow copy of nested list.

Deep Copy :-

import copy

list1 = [1,2,3,4]

list2 = copy.deepcopy(list1)

When we have a simple list then this deep copy is same like shallow copy.

list2[1] = 100

list2

[1,100,3,4]

list1

[1,2,3,4]

list1 = [[1,2,3], [3,4,5], [5,6,7]]

list2 = copy.deepcopy(list1)

list2[1][0] = 100

list2

[[1,2,3], [100,4,5], [5,6,7]]

list1

[[1,2,3], [3,4,5], [5,6,7]]