10/05/2022

PYTHON PROGRAMMING:

Guido Van Rossum.

- Procedural & embedded.
- Object oriented programming (features).

* Very simple syntax
* Indentation (way we write our code) is very crucial in python.
  - Position of a sequence of statements should be written in a specific way.
* Expressive language: few lines are enough to print a simple code
* Uses interpreter: which compiles & executes each and every line; line by line.
  If it encounters any error while compiling; it stops at a specific location; Python is portable.
* Python is an interpreted language which makes debugging of the code very easy.
* Dynamic memory allocation.

→ OBJECT ORIENTED PROGRAMMING:

1) Identifies the objects required for a particular code.
2) Thinks of different sub-topics for each object.
3) Thinks of different operations to be performed on object.

- Construct a class regarding with objects & diff. attributes.

* We can use different other languages in python as well

* Large number of libraries are available which are readily being used.

* Open source: Accessable to everyone: developing codes and can be used uploaded in this open source which can be used by others.

* Modes:
  - Interactive Mode: Giving instruction & it executes; file is not opened.
  - Script (file) Mode: file is opened and code is saved in it.

* Extension: filename.py

→ Output is seen in interactive mode.

* Not going to mention datatype & variable need not be declared earlier. It allocates space directly

```
>>> marks=25.5
>>> name="Haa"
>>> print(marks)
25.5
>>> print(name)
Haa
```

```
>>> print(marks, "\n", name)
25.5
Haa
>>> print( marks, name)
25.5 Haa
```

* print ("Hello") | print ("""Hello""")
  print ('Hello')

* '#' symbol is used for commenting → Single line (Generally)
* (or) we can use """ Statements """ → Multi line comments.

*
```
l=10          | l, b = 10,30
b=30
print ("area of rectangle=", l*b)
print (l); ⇒ print its value
print ("breadth=", b)
```

⇒ File Mode

```
output
10
breadth=30
```

>>> ⇒ python prompt.

11/05/2022

IDLE: Interactive Development & learning Environment.

Python is a high-level language (HLL) and is case sensitive.
- Secured language.

Class contains attributes as well as functions.

Strings, lists, Numpi, Topple,

Object oriented programs supported by PYTHON: or any HLL
- class, object, inheritance, polymorphism,

encapsulation, abstraction

* Python is portable ~~means~~ because it is converted to byte code (code is not depending on the machine) and can be run on any hardware as well as software.

* Embedded: Code written in python can be embedded in other languages like C, C++, Java etc. and vice versa.

* Python is extensible due to addition of diff. modules.

* Python code is versatile in nature means we can use it for any kind of applications in diff. domains.

* Python is a slow due its interpreter and it is not efficient for large amount of code.

* Read and print data:

```
>>> print ("Hello") ↵
Hello
>>> l = "World"
>>> print ("Hello", l)
```

Hello World

* >>> a = 10
  >>> print(a)
  >>> b = 20
  >>> print(a, b)

>>> print("Hello\n World")
Hello
World.
>>> print("Hi \t Hello")
Hi Hello.

>>> print("ab.....\
xyz")

→ allows to continue the string in next line.

>>> a = 10
>>> print(a)

(or) >>> a = 10 ; print(a).

Mandatory.

>>> print("Hi \\ Hello")

Hi \ Hello.

* No character datatype in python.

* input ⇒ to read the data from keyboard.

>>> name = input()
Raj
>>> print(name)
Raj

In shell.
>>> python fname.py ↵
⇒ executes.

file.py

Run.

```
name = input()
print(name)
```

```
>>> name=input("Enter name:")
    Enter name
```

* To determine the type of given input:

```
a=5
>>> type(a)
<class: 'int'>
```

The input taken from the keyboard is always of string form.

*

file.py

Run

```
no=int(input("Enter name"))
no=no+5
print(no)
```

⇒ OPERATORS:-

* Arithmatic Operators: +, *, /, %, **, // → floor divi
  ↓ exponent

```
>>> print (5//4)
    1
>>> print (5/4)
    1.25.
```

No conditional operators in Python.

* Relational / Comparison Operators:

>, <, >=, ==, <=, !=.

** Relational operators gives a boolean output.

True = True is only printed.

```
>>> a=100
>>> b=20
>>> print(a>=b)
>>> print(a<=b
```

output

True

False.

*Logical Operators: and, or, not are used instead of &&, ||, !.

```
*  a=10
   b=20
   c=30
   print((a>b) and (a>c))
```

false false.

false.

* print(not(a>c))

True.

*Bitwise operators: &, |, ^, <<, >>, ~

Exclusive or: if we have odd no. of ones it returns 1

```
  0101
  1000
 ------
  1001
 ------
```

*Left shift & right shift

```
  00011010
     << 1
  00110100.
```

* Assignment Operators :

```
>>> a = 10
>>> b = 20
>>> a = b
>>> print (a)
    20
```

$x = y = 10$

$x = 10,\ y = 10$

* In-place assignment operators:

+=, *=, /=, **=, //=, -=

* rno, age, cgpa, name = 25, 13, 9.9, "xyz"

12/05/2022

DECISION CONTROL STATEMENTS:

It is a statement that determines the control flow of a set of instructions (it decides the sequence in which the program is to be executed)

→ Sequential Control Statements
→ Selection / Conditional control statements
  - if
  - if else
  - Nested if
  - if elif else.
→ Loop / Iterative C.S.
  - for
  - while

* Write a program to swap the contents of given 2 variables.

→
```
no1=25
no2 =30
print ("Before swapping", no1, no2)
t=no1
no1=no2
no2 = t
print("After Swapping")
print("no1=", no1, "no2=", no2)
```

also
no1, no2 = no2, no1

* Write a program to calculate the area of circle given its radius.

~~print("Enter the radius")~~

```
r= int(input("Enter radius"))
area = 3.14 * r * r
print("Area =", area)
```

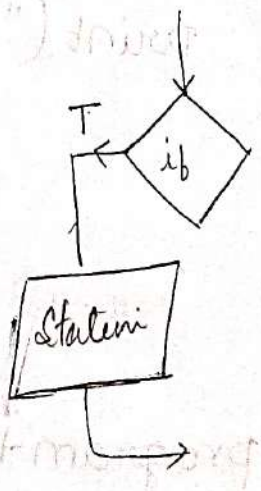
* Selection Control Statements :

~~if(exp/cond):~~

1) if statement :

```
if (exp/cond):
    statement 1
    statement 2
    statement 3
```

4 spaces.



* Write a program to check whether a candidate is eligible to vote.

```
age = int(input("Enter age"))
if age >= 18 :
    print("Eligible to vote")
```

2) If else statement:

```
if exp/cond :
    statement block.
else :
    statement block
statements.
```

```
* age = int(input("Enter age"))
  if age>=18 :
      print("Eligible to vote")
  else :
      print("Wait for", 18-age, "years")
```

* isalpha
  isdigit
  isspace.
  check for type of character

* Write a program to check if given no. is even (or) odd.

```
a = int(input("Enter no"))
if a%2==0:
    print("Even no")
else:
    print("Odd no")
```

3) Nested if Statement:

```
if cond1:
    if cond2:
        statement block.
    else:
        statement block.
else:
    statement block.
```

* Write a program to print the smallest among the given 3 integers.

```
print("Enter 3 numbers")
a = int(input())
b = int(input())
c = int(input())
if a<b:
    if a<c:
        print("a is smallest")
    else:
```

```
if a>b:
    if c>b:
        print("b is smallest")
    else:
        print("c is smallest")
else:
    if c>a:
        print("a is smallest")
    else:
        print("c is smallest")
```

4) if-elif-else statement:

```
if (test exp1):
    statement block
elif (test exp2):
    statement block
elif (test exp3):
    statement block
else:
    statement block
```

*
```
if ((a<b) and (a<c)):
    print("a is smallest")
elif ((b<a) and (b<c)):
    print("b is smallest")
```

```
else:
    print("C is smallest")
```

* Loop Control Statements:

1) While loop:

```
while(cond):
    statements block.
    =
    =
    =
else:
    print("Done").
```

⟹ If condition fails else part is executed

* Write a program to find the sum of first 10 natural numbers.

```
i=1.
s=0
while (i<=10):
    s=s+i
    i=i+1
print("Sum of natural nos", s)
```

2) for loop:

```
for loopvar in sequence:
    statement blocks.
```

1) range(1, 5) → first, last — only last-1 are printed

2) range(1,5,2) → step; tells the increments by how much.

range function

starting value, last value.

```
for i in range(1,5):
    print(i)
```

last value − 1 will only be printed

3) range(6): initial will be 0
↳ last value

only last-1 is printed.

* for i in range(6):
    print(i, end=",")

output
0,1,2,3,4,5,

* for i in "Hello":
    print(i)

We can use else in for loop.

SYNTAX:

range(beginning, end, [step])

for loop in const var in seq:
    statements
else:
    statements.

*Write a program to calculate the factorial of a number.

```
a = int(input("Enter no"))
f = 1
for i in range(1, a+1)
    f = f * i
print(a, "factorial is", f)
```

12/05/2022 WEEK-1

1) Write a program to display your Biodata

2) Write a program to implement the basic arithmatic operations. (Interactive mode)

3) Write a program to check whether the triangle is equilateral / scalene / isosceles.

```
a=10
b=-a
print(b).
```

4) Write a program to check whether the given number is +ve /-ve/0.

5) Write a program to calculate the area of triangle using Heron's formula.

6) Find the greatest among the 3 nos.

7) WAP to enter the marks of student in 4 subjects; Calculate total, and display the grade obtained by the student. If the student scores % | above 75% = distinction
>=60% & <75% = 1st division
>=50% & <60% = 2nd division
>=40% & <50% = 3rd division
<40% = fail

8) WAP, given no is palindrome or not. Whether

9) WAP to display multiplication table of a given no.

10) WAP to check given no is prime or not.

11) Display even & odd upto the given limit.

12) WAP to display the sum of series

$1^2 + 2^2 + \cdots\cdots + n^2$

13) WAP to find sum of digits of given no.

14) Display armstrong number in the given range.

15) WAP to calculate GCD & LCM.

16) Display even nos in reverse order from 50 to 10

17) WAP to calculate the total amount of money in piggy bank ₹10, ₹5, ₹2, ₹1..

13/05/2022

The input taken from the keyboard: it is default a string.

* **Break** Used only in loops.

Syntax: break

It helps us to comes out of the loop skipping even else part.

* Break is used to terminate the execution of nearest completing loop in which it is used.

SYNTAX:

```
for ≡
    for
        if cond:
        break.
```

* Write a program to check whether given no is prime or composite.

```
b= int(input("Enter a no"))
for i in range(2, b//2):
    if(b%i==0)
    print("Prime number")
    print("Prime number")
    flag = 1
```

```
b = int(input("Enter a no"))
for i in range(2, b):
    if(b%i==0)
        flag = 1 → break

if(flag==0):
    print("Prime No")
else:
    print("Composite No").
```

*
```
i = 1
while i<=10:
    print(i, end=" ")
    if i==5:
        break
    i=i+1
print("\n Done").
```

O/P:

1 2 3 4 5

Done.

~~* for i in range~~

* CONTINUE: Used only in loops.

SYNTAX: continue

```
while cond:
    ≡
    continue
    ≡
print
```

Skip

When the compiler encounters continue statement in for (or) while loop; the rest of the statements are skipped; and control is transferred to the testing condition of the nearest enclosing loop.

```
* for i in range(1,11):
      if (i==5):
          continue
      print(i, end=" ")
  print("\n Done")
```

O/P:

1 2 3 4 6 7 8 9 10

Done.

```
* while ( ):
      if cond:
          continue
```

```
* for ( )
      if cond:
          continue
```

```
* for ( )
      for ( )
          if cond:
              continue
```

```
* for x in range(6):
      if x==3: break
      print x
  else:
      print("Finally finished")
```

O/P: 0
1
2

* PASS STATEMENT:

syntax: pass.

```
if a>0:
    print(a)
else:
    pass.
```

* The pass statement is used when a statement is required syntatically but when no code is required to be executed (as a place holder). It specifies a null operation.

```
for i in "Hello":
    pass.
    print i
print("Done")
```

O/P:

H
e
l
l
o
Done.

* for i in "Python":
    error.

```
for i in "python":
    pass.
```

** NESTED LOOPS:

```
for i in range(1, 4)
    for j in range(1, 6)
        print(j, end=" ")
print() → goes to new line.
```

1 2 3 4 5
1 2 3 4 5
1 2 3 4 5

*Nested loops:- loop inside a loop.

Q) WAP to print
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5

```
for i in range(1,4):  ---> no.of itrations
    for j in range(1,6):  ---> values to be printed
        print(j, end=" ")
print("\n")  ---> (or) print()  ---> prints new line.
```

Q) WAP to calculate roots of quadratic eqn.

```
a = int(input("Enter coeff of x^2"))
b = int(input("Enter coeff of x"))
c = int(input("Enter const value"))
d = ((b**2) - (4*a*c))
If d==0:
    r1 = ((-b)/(2*a))
    print("root=", r)
elif d>0:
    r1 = (((-b + (d**0.5))/(2*a))
    r2 = (((-b - (d**0.5))/(2*a))
    print("roots are=", r1, ",", r2)
else:
    print("root doesnot exist")
```

* FUNCTIONS: Set of instructions which perform a specific task.

Syntax: def funcname (v1, v2, v3):

Function defn:
- fn header: documentation string (tells what the fn does.)
- fn body: function statements, return [exp]

01/06/2022.

* Advantages:
1) Reuse
2) Readability / Length of code decreases.
3) Save time
4) Debug easily.

* Documentation string can be written in multiple lines.

* Function code will be executed only when it is called.

Ex:
```
def display():
    """ This function displays Hello msg """
    print("hello")
```
return ⇒ not mandatory.

Syntax: function call

funcname (v1, v2 ..... vn) → if variables are to be specified.

no fn declaration

* Write a function to subtract 2 integers.

```
print("Enter 2 numbers")
a = int(input())
b = int(input())
s = diff(a, b)        → actual parameters
print("Diff. of 2 numbers = ", s)
```

```
def diff(x, y):        # formal parameters
    return x-y
```

- print(diff(a+10, b))
- print(diff(100, 20))

* The count & order, type of the parameters passed to a function should be same in function defn and call.

* Ex:
```
print("Hi")
def display():
    """The fn displays Hello"""
    print("hello")
print(display())
print(display.__doc__)
```

OUTPUT:
```
Hi
Hello
None
```

→ Syntax for ~~do~~ to print documentation string:
```
print(funcname.__doc__
```

* Write a function to display "Vasavi college of Engineering" 100 times

```
def display():
    """Displays Vasavi College of Engineering"""
    print("Vasavi College of Engineering")
```

return

```
for i in range (0, 100):
        display()
```

*Assigning fn name to another variable:

```
a = 50
b = 100
diff(a,b)
op = diff
op(a,b)
```

*SCOPE:

⇒Local and global scope variables:

Scope of a variable is the portion of the program where the variable is recognised.

The life time of a variable is a period throughout which variable exists in the memory.

```
def fun():
    x = 10
    print("Inside function =", x)

x = 20
fun()
print("Outside function =", x)
```

If a local & global variable have same name; within a function local variable

will have domination over global

*
```
n1=10
print(n1)
def fun(n2):
    print(n2)
    n3=30
    print(n3)
fun(20)
print(n1)
print(n3)
```

OUTPUT:

```
10
20
30
10
error
```

*
```
n1=10
print(n1)
def fun(n2):
    n1=n2+10
    print(n1)
```

} error ∵ Global variable cannot be modified

instead:

```
def fun(n2):
    global n1
    n1=n2+10
    print(n1)
fun(20)
print(n1)
```

OUTPUT:

```
30
30
```

*
```
def show():
    global var1
    var1="Hi"
    print(var1)
```

OUTPUT:

```
Hi
Hi
```

show ()
print (vars)

- We can define one function inside another function.

Ex:

```
def o_fun():
    o_var=10
    def i_fun():
        i_var=20
        print (o_var)
        print (i_var)
    i_fun()
    print (o_var)
    print (i_var)
o_fun()
```

OUTPUT:

10
20
10
error

02/06/2022

* TYPES OF ARGUMENTS:

1) Required arguments | Positional arguments
2) Keyword arguments
3) Default arguments
4) Variable length arguments

* Required arguments

1) WAP to calculate cube of a given no:

```
def cube(x):
    return (x*x*x)
```

```
n = int(input())
cube(n)
print("cube=", cube(n))
```

* Keyword arguments:

Ex:
```
def display(name, age):
    print("Name=", name)
    print("Age=", age)
```

- display(age=15, name='xyz')
- display(name='ABC', age=10).

} order do not matter

display("Raj", age=20);

display(name="A

↓

not acceptable

* Keyword arguments makes the code easier to read & understand.

* The values are not assigned to arguments acc to their portion but bound in name/keyword

* Having a keyword argument after requires argument; it causes error.

* Default Arguments:

```
def display(name, course="CSE"):
    print("Name=", name)
    print("Course=", course)

display("Rajeev", "CSE")
display("Shivani")
display("Tej", "IT")
```

⇒ Always the default arguments passed should be at the end only.

* Default arguments are the values that are provided while defining the call.
* During the function call, if a value is passed, it will overwrite the default value.
* There is no restriction for no. of default arguments

⇒ Variable Length Arguments:

* Write a function to display the Semester and the courses offered during that Semester

```
* def display(sem, *sub):
      print("sem=", sem)
      for i in sub:
          print(i)

display("IT", "LST", "PP", "M", "AC")
display("IEE", "PPS", "SPOD", "BEM", "BED", "C
```

→ variable length arguments } Should be at end

Output:

```
SEM-II
LST
IT
LST
PP
M
AC
SEM-I
IEE
PPS
SPOD
BEM
BED
CLA
```

* Default argument should be after non-default arguments and also before variable length arguments.

* ' is placed before the variable name that holds the values of non-keyword variable argument.

TYPES OF FUNCTIONS:

Built-in functions
User-defined functions.

> format() function:

Ex: format(3.1416724, ".2f") - OUTPUT: 3.14

format(1234567, ',') → 1,234,567

*SYNTAX:

format(value, format specifier)

format("Hello", '<10') → Hello_____ (Left Justified)

format("Hello", '>30') → _____Hello (Right Justified)

format("Hello", '^30') → __Hello___

* """Welcome to
"CSE"
department""" } Display everything b/w 3 single quotes

* "Welcome to \
CSE"

Output:
Welcome to CSE

* print("python\n programming")

print("python\' programming")

print(R,"python\n program")

↑ Raw String

OUTPUT: Python
Programming

'Python' programming

python \n programming

03/06/2022

* Recursive Functions :

- In certain cases of some complicated codes; writing recursive functions helps us to read & understand the code easily.

```
def fact(n):
    if (n==0 (or) n==1):
        return 1
    else:
        return (n* fact(n-1))
```

* Write a program to calculate the exponent of given no.

⇒

```
def exp(x, n):
    if (n==0):
        return 1
    elif (n==1):
        return x
    else:
        return (x* exp(x, n-1))

x = int(input("Enter a number"))
n= int(input("Enter the exponent"))
```

r=exp(x, n); print(r)

→ Recursive function is time consuming and takes a lot CPU time.

* Write a program to calculate gcd of 2 numbers:

```
def gcd(n1, n2):
    if (n1 % n2 == 0):
        return n2;
    else:
        return gcd(n2, n1 % n2)

a = int(input("Enter a number"))
b = int(input("Enter a number"))
print(gcd(a, b))
```

* Write a program to calculate the sum of digits using recursive function:

```
def s(n):
    if (n==
    rem = n%10
```

```
def s(n):
    if (n==0):
        return 0
    else:
        return (n%10 + s(n//10))
n = int(input("Enter a no"))
print(s(n))
```

*DATATYPES:

- Tells us the type of data we should enter
- Specifies the range of given datatype and helps us to enter specified numbers only.
- Helps in allocating space.

1) Numbers → integer, float
2) String
3) Tuple
4) dist
5) Dictionary

* print ("Hello" + "World") ⇒ Concatenation!
  O/P: Helloworld.

* print ("Hello" + "5") | print ("Hello" + 5)
  O/P: Hello5. | error.

* print ("Hello" *3) ⇒ Repetition
  O/P: HelloHelloHello

*TUPPLE: a list of data in parantheses.

⇒ It is not mandatory that data in list is of same datatypes. (Mix of datatypes)

⇒ Content in a tupple cannot be edited.

Ex: a = (10, 20, "Sivani")

We can repeat the values.

*List: a list of data in square brackets

- Data can be of same (or) different datatypes

```
l=[10,20,30]
print(l)
```
O/P: 10,20,30

* We can repeat the values.

```
print(l[0])
```
O/P: 10

```
print(l[0:2])
```
O/P: 10,20

↳ Slicing

```
print(type([10,20,30]))
```
O/P: <class 'tuple'>

```
print(type("Hello"))
```
O/P: <class 'str'>

*Dictionary: {"apple":"fruit",}

KEY-VALUE PAIR

key ↙ ↘ value

```
dict={1:"one",}
print(dict[1])
```
O/P: one

- 2 keys should never be same
- But 2 values can be same.

*TYPE-CONVERSION:

implicit type conversions ⇒ automatic
explicit type conversions ⇒ we need to specify.

int(x)	list(x)	chr(x)
float(x)	set(x)	dict(x)
long(x)	oct(x)	
str(x)	set(x)	
tuple(x)	ord(x)	

↳ converts a single char to int

To know about any type or functions in python: help(int)
(or) help(range)

* int(2.90) = 2
round(2.90) = 3
round(2.90324135) = 2.93

MODULE: ⇒ Built-in modules.

- consists of mostly used functions which are defined previously and saved as python files.

We can use a pre-defined modules we should use: IMPORT

Syntax: import mname.

- Modules can also be user-defined.

Ex:
```
import math
print(pi).
```

- It is inefficient to import the whole set of code from the modules.
We thereby use (from) tag to import the specified function.

~~from import statement~~.

* from filename import fn name;

Eg: from math import pi, sqrt.

```
print(pi).
print(sqrt(25))
```

04/06/2022

* LAMBDA FUNCTIONS: → No function name.

SYNTAX:

lambda arguments : expression.

Ex: Add = lambda x, y : x+y

```
print(Add(2,3))
```

⇒ We cannot access these lambda functions in any other function.

⇒ In lambda function; we can use only one expression at a time.

*WAP to find the smallest no. among the given 2 numbers:

* def smaller(a, b):
    if (a<b):
        return a
    else:
        return b

* s= lambda x, y: x+y
  d= lambda x, y: x-y
  print("Smallest no is", smaller(sum(-2,-3), diff(-1,2)))

* Write a lambda expression to find the sum of first 10 natural nos.

S= lambda : sum(range(1,11))
print(S())

* Write a program to demonstrate lambda function call from another lambda function.

add = lambda x,y: x+y
multiply_and_add = lambda x,y,z: x * add(y,z)
print(multiply_and_add(2,3,4))

O/P: 14

MODULES:

⇒ User defined modules:

- Ex: calc.py

```
def add(x,y):
    return (x+y)
str= "message"
```

def sub(x,y):

⇒ pgm.py return (x-y)

```
import calc
print (calc.add(2,3))
print (calc.str)
```

This helps us to avoid the ambiguity if 2 files have same functions

o/p: 5

checks in
current director
python paths
lib directory
else it shows an error

To avoid ambiguity

```
from calc import add as addition
print (addition (2,3))
print (add(3,4)) ×
```

from calc import * : imports whole file.

*private variables: `__` secured ; ∴ they are not imported into another function.

all public variables can be imported.

*Module is a file with .py extension that has definitions of all functions & variables that can be used in other programs.

*The other program should import that particular module.

To know the module name.

```
print(__name__)
```

* Built-in modules:

→ **sys:** standard library module.

Ex:

```
import sys
print("Hello")
sys.exit(0)
print("end")
```

O/P:

```
Hello
```

sys exit(arg)

→ 0-127 numbers and any other arguments.

Min: 28 bytes

```
print(sys.getsizeof(2))
```

Each integer i.e. 2, 200, etc will have a different size

We can give any type of data to getsizeof function.

Sys.path: To know the path given in Python paths

* import sys

```
print(len(sys.argv))
print(sys.argv[0])
```

⇒ default 1st argument will be the program name.

```
for i in range(len(sys.argv)):
    print(sys.argv[i])
```

* Write a program to add the given numbers through command line.

```
import sys
for i in range(1,len(sys.argv)):
        s=s+int(sys.argv[i])
print("Sum=",s)
```

→ Math module (math):

```
print(math.pi)
print(math.sqrt(2))
```

Ex: Sin
Cos
Tan
Ceil

```
from math import ceil
print(ceil(2.91))
```

* Random: Gives a random no in given range.

```
import random
for i in range(10):-
    value = random.randint(1,100)
    print(value)
```

* Write a program to add the given numbers through command line.

```
import sys
for i in range(1,len(sys.argv)):
    s=s+int(sys.argv[i])
print("Sum=",s)
```

→ Math module (math):

```
print(math.pi)
print(math.sqrt(2))
from math import ceil
print(ceil(2.91))
```

ex: Sin
Cos
Tan
Ceil

* Random: Gives a random no in given range.

```
import random
for i in range(10):
    value=random.randint(1,100)
    print(value)
```

07/06/2022

* Time:

```
import time
print(time.time())
```

↳ prints seconds passed since Jan 1 1970.

```
time.struct_time(tm_year= ,tm_mon= ,
    tm_mday= , tm_hour= ,
    tm_min= , tm_sec= , tm_wday=
```

tm_isdst =)
takes 0, 1, -1

* t = time.time()
  ltime = time.ctime(t)
  print(ltime)

O/P: Tue Jun 7 2:45:29 2022

* import calendar
  print(calendar.month(2022, 6))

Package: Consists of modules which are created to perform a single task and also contains packages.

__init__.py ⇒ default file

If a package contains the above (empty file) is called package & it is different from a normal directory.

→ A package is a hierarchical file directory structure that has modules and other packages within it.

- Every package is a directory which must have a special file called __init__.py

MyPackage:
- Modules
  - fun1
  - fun2
- Module 2
  - fun3
  - fun4

→ It is simply added to indicate that this directory is not an ordinary directory and contains a python package.

⇒ SYNTAX:

import mypackage.mymodule

(or)

from mypackage import mymodule

* Globals() : print(globals)
  locals() : print(locals())
  reload()

→ Globals(): returns the list of global variables used in the code.

→ locals(): returns the list of local variables in which it is being used.

→ reload(): used to reload a module whenever it is specified in other function

---

* STRINGS:

To continue a string in a new line, we need to end with a '\'

```
str1 = "Hello"
str2 = "World"
str3 = str1+str2
```

→ CONCATENATION

```
print(str3)
str4 = str+4 x → error
```

O/P: Helloworld

str5 = str1 + "4" Hello 4

str6 = str1 + str(4) Hello 4

```
str1 += str2
print(str1)
```

appending

O/P: Helloworld.

* To refer each character of a string:
We can use index to access those.

```
print(str1[0])
print(str2[0])
```

⇒ Index can be either an integer or an expression that evaluates to an integer.

*
```
msg = "Hello!"
index = 0
for i in msg:
    print("msg[", index, "] = ", i)
    index += 1
```

O/P:

msg[0] = H
msg[1] = e
msg[2] = l
msg[3] = l
msg[4] = o
msg[5] = !

*
```
str1 = "Hello, World"
i = 2
print(str1[i])
print(str1[i*3+1])
```

O/P:

l
W

```
def display():
    print("Hello")
display()
def display():
    print("CSE")
display()
```

```
* str1 = "Hello"
print(str1, end=' ')
print("world")
```

Hello World

```
* str1 = "Hello"
  print(str1)
  print("world")
```

O/P:
Hello
world.

08/06/2022

→ Strings are immutable i.e. cannot be modified.

* id() → gives the address of the 1st byte of a string

```
str1 = "Hello"
print(id(str1))
str1 = str1 + "world"
print(id(str1))
```

∴ str1 = "Hello"
str1[1] = 'a' } ✗

```
str3 = str1
print(str3)
print(id(str3))
```

} Separate/same memory address for str1 and str3.

* STRING FORMATTING OPERATOR:

Syntax:

"<format>" % (<values>)

Eg: name = "Ram"
age = 9

print ("Name = %s and age = %d" % (name

⇒

%u = unsigned integer

%o = octal number

%x = hexadecimal number

* STRING METHODS & FUNCTIONS :

* Methods are the functions which can be called only over an object. Whereas a function is called directly.

Ex: str = "Hello"
print(str.lower())
print(str.upper())
} Methods ⇒ functions are called over an object.

O/P: hello
HELLO

* print (str.capitalize()) O/P: Hello
* print(str.center(10, '*')) O/P: **Hello***
↓ width of whole string
* print(str.endswith("lo", 0, len(str))
* print(str.startswith("H", b, end, len(str))
* print(str.find("e", 0, len(str)) ⇒ beginning

gives -1 if given character is not in the string otherwise there is no error & continues the code.

* print(str.index("a", 0, len(str)))
gives an exception (error) if character is not in the string

rfind: to check from end for a character

* str1 = "ABC123"
print(str1.isalnum()) ⇒ T
print(str1.ispace()) ⇒ F
print(str1.isalpha()) ⇒ F

⇒ lstrip(str1) } Functions
Hello
rstrip(str1). }

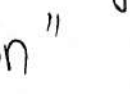
max("Hello") } ⇒ Ascii Values.
min("Hello") }

* str="he"
msg="hello world hellohello"
print(msg.count(str, 0, len(msg))) : 3
str.isdigit
print(msg.replace("he", "Fo"))

* SLICE OPERATION:

A substring of a string is called slice
[ ] → slicing operator.

str="python"



-6 -5 -4 -3 -2 -1
python
0 1 2 3 4 5

Syntax: str[start : end]

str[:5] ⇒ Pytho

str[:] ⇒ python
str[1:] ⇒ ython

str[2:5] ⇒ tho

str[-1] ⇒ n

str[-2:] ⇒ on

str[0:-2] ⇒ pyth

str[-5:-2] ⇒ ...

* Slicing will take place only from left to right.

10/06/2022

* Stride: s1 = "python"

s1[0:5:2] → pto

↳ species the index of next retrieving character of a string.

⇒ Specifying stride while slicing strings:

→ The third argument, stride refers to the number of characters to move forward after the first character is retrieved from the string.

→ The default value of stride character is (1).

s1[::-1] ⇒ prints the string in reverse order

O/P: nohtyp

* ord(): Returns the ascii value of the string

s1 = "A"

print(ord(s1)) ⇒ 65

* chr(): Returns the string for a given value

(89)

print (chr(65)) ⇒ A.

⇒ Operators: In and Not in

It is used to check whether the string is present in other string (or) a character in a word.

* Ex: str1 = "Welcome to Python"

```
str2 = "to"notin
if str2 in str1:
    print("found")
else:
    print("nOt found")
```

O/P: found

Not found

⇒ >>>'p' in "python" | >>>"word" in "word"
True | True

>>>'r' in "python"
False.

⇒ Comparing Strings: <, <=, >, >=, ==, !=

Ex: "to" == "To" → False

"to" > "welcome" → False.

"to" == "to" → True.

⇒ Iterating Strings: S1 = "Welcome to python"

```
for i in S1:        # i → character
    print(i, end='')
```

O/P: welcome to python

Ex: msg = "welcome" ; index = 0

```
while index < len(msg):
    letter = msg[i]
    print(letter, end=' ')
    index += 1
```

* WAP to copy a string unto another string using copy function.

```
s1 = " Vasavi "
s2 = "College"
copy(s1, s2)
```

~~for i in s2:~~

~~s2(len(s1)+i) = s1~~

~~s1 = s1 + i~~

~~print~~

```
def copy(s1, s2):
    for i in s2:
        s1 = s1 + i
    print(s1)
```

```
def copy(s1, s2):
    while i < len(s2)
        letter = s2[i]
        s1 = s1 + letter
        i += 1
    print(s1)
```

*String Module:

import string → constants.
→ functions
→ variables.

* • string.ascii_letters
  • string.ascii_lowercase
  • string.ascii_uppercase
  • string.digits
  • string.hexadigits
  • string.lowercase
  • string.octdigits
  • string.punctuation
  • string.printable
    ↳ escape characters not printed.

* import string
  print(string.digits)
  O/P: 0123456789

* print(string.ascii_letters)
  O/P: abcdef-----zABCD--Z

* print('g' in string.lowe
  → True

* str = "Welcome to the world of python"
  print("Uppercase=", str.upper())
  print("Lowercase=", str.lower())
  print("Split=", str.split())
  print("Join=", '-'.join(str_split()))
  print("Replace=", str.replace("Python", "Java"))
  print("Count of O=", str.count('o'))
  print("Find of=", str.find("of"))

15/06/2022

* Regular expressions: string manipulation

[_, a-z, A-Z] & [a-z, A-Z, 0-9_]

Specify the pattern using regular expressions. To check if a pattern is present in a given string by using a module re.

Eg: "welcome"

"Welcome to python"

```
import re
pattern = "welcome"
str1 = "Welcome to python"
if re.match(pattern, str1):
    print("pattern found")
else:
    print("Not found")
```

pattern = "Pyth

* Match function:

*SYNTAX: checks for a given pattern from beginning of string

match(pattern, string, flags=0)

* if it is matching → return match object.

If the pattern do not match at the beginning, it will return false.

* if it doesn't match → returns none.

match object contains starting index, ending index a Span.

* Search function checks for a given pattern anywhere in the string.

⇒ search(pattern, string, flags=0)

⇒ A regular expression is a special sequence of characters that helps to match or find strings in another string. These are a special text string i.e. used for describing a search pattern to extract information from text such as code, file or any other document.

⇒ If re.match finds a match it returns match object and none otherwise.

re.match finds a match only at the beginning of the string.

⇒ If "search function" searches for a pattern anywhere in the string. It searches for the first occurence in the string and returns flags.

* Sub function:

re.sub (pattern, repl, string, max=0)

Sub functions replaces all the occurences of pattern in string with repl substituting all occurences unless any max value is provided.

Eg: import re

s1 = "She sells sea shells on the sea shore"

p1 = "sea"

repl = "ocean"

newstring = re.sub(p1, repl, p2, s1)

print(newstring).

⇒ O/P: She sells ocean shells on the ocean

newstring = re.sub(p1, repl, s1, 1).

print(newstring)

⇒ O/P: She sells oceanshells on the sea shore

* findall function:

SYNTAX:

re.findall(pattern, inputstring, flags=0).

Returns all the occurences of a particular pattern in given string.

```
import re
s1 = "She sells sea shells on the sea shore"
p1 = "sea"
l = re.findall(p1, s1)
print(l)
```

O/P: ['sea', 'sea']

```
s1 = "s13, s51 sea shells"
p1 = "[a-z][0-9]*"
l = re.findall(p1, s1), print(l)
```

O/P:

['s13', 's51']

⇒ findall function is used to search and return a list of matches of the pattern in the string

* finditer():

SYNTAX:

mlist = re.finditer(pattern, inputstring, flag=0)

```
import re
s1 = "s32 a15 ∞"
p1 = "[a-z][0-9]*"
m = re.finditer(p1,s1).
for i in m:
    print(i.start())
    print(i.end())
    print(i.span())
```

[0-9]{9}

\d → represents the digits.

[\d]{9}

4
6
[4,6]

0
2
[0,2]

⇒ Meta Characters:

. → any symbol.

(re)* → * : 0 (or) more occurences.

(.)* → . can occur any no. of times.

(re)+ → + : 1 (or) more occurences.

[xyz]* ⇒ searching/matching with these characters only.

→ any no. of occurences of the given characters

[^xyz] ↔ trying to search for characters other than these characters.

\A(or)^Hi → Searching for pattern beginning with Hi

\Z(or)Hi$ → Searching for pattern ending with Hi

* re? → 0(or)1 occurence
* [0123456789] (or) [0-9] ⇒ searches for a digit. → [\d]
* [a-z] → search for a single lowercase letters.
* [0-9]* → all the patterns with digits are considered.
* [a-z][A-Z] → for both lower & upper case letters
* [a-z][0-9] → for both lowercase letters & digits.
* Pattern can be of form:
  - simple string
  - metacharacters
  - character class

17/06/2022

* Character Class: Set of characters which constitute a class.

  eg: [aeiou] ; [0-9] ; [0-9][aeiou] ; [a-f] ; [b-g]

⇒ WAP to check whether the given identifier is valid in python or not

```
ide = input("Enter the identifier")
p = "[a-z A-Z _][a-z A-Z _ 0-9]*"
if re.search(p, ide):
    print("Valid Identifier")
else:
    print("Not valid")
```

[7-9][0-9]{9} r"\d{2}-\d{2}-\d{1}

* [re]{c} ⇒ regular expressions repeats for c times.

[re]{c,} ⇒ re can repeats more than c times.

* for words : \w
non-character words : \W
for spaces : \s
non-spaces : \S

```
* import re
  p1 = r"[aeiou]"
  if re.search(p1, "apple"):
      print("Match apple")
  else:
      print("Not Match")
  p2 = r"hi(hello)*"
  if re.search(p2, "hihellohellohello"):
      print("Matched")
  else:
      print("Not Match")
```

* Output:
Match apple
Matched.

* ^pr.y$: Starts with 'pr' and ends with 'y'

* LIST: Collection of data elements;
- compound datatypes

SYNTAX: listname = [data elements]

```
          0  1  2  3  4
Eg: l1 = [1, 2, 3, 4, 5]
    l2 = ["a", "e", "i", "o", "u"]
    l3 = [1, "a", "@", "hello"]
```

- Lists are mutable i.e. lists can be changed
- l[ ] ⇒ empty list

```
print(l3[3])
```
O/P: hello

* To access the values of the list, we use slice operation:

```
l4 = l2[ : : 2]
print(l4)
```
O/P: ['a', 'i', 'u']

```
l5 = l2[ : ]
print(l5)
```
O/P: ['a', 'e', 'i', 'o', 'u']

```
l5[2] = 100
print(l5)
print(l2)
```
['a', 'e', 100, 'o', 'u']
['a', 'e', 'i', 'o', 'u']

* l6 = l2 : The values of l2 are assigned to l6 due to which if we change the value of l2 ; changes are observed in l6 and vice versa.

* l5 = l2[:] : Values of l2 are copied to l5 ; i.e. if we change values of l2 changes are not observed in l5 and vice versa.

* print(max(l1)) — O/P: 5
  print(min(l2)) — O/P: 'a'
  print(len(l2)) — O/P: 5
  print(sorted(l2)) — O/P: 1 2 3 4 5
  ↳ sort the elements.

  ascii values of d < u < l

O/P:

* l1 = [1, 2, 3, 4, 5]

→ del l1[2:4]
  print(l1) — O/P: [1, 2, 5]

→ del l1[:]
  print(l1) — O/P: [ ]

→ del l1
  print(l1) — O/P: Error

* l1[1] = [10, 20, 30]
  print(l1)

O/P: [1, [10, 20, 30], 3, 4, 5]

  print(l1[1][0])

O/P: 10

print(l1[-4:])

O/P [3, 4, 5]

- A list can be concatenated with another list

```
l = [1, "one"] + [2, "two"]
print(l)
```

O/P: [1, "one", 2, "two"]

*list is a versatile datatype.
*It is a sequence in which elements are returned as a list of ; separated values b/w square brackets.
*It can have elements that belong to different datatypes.

```
list_variable = [var1, var2, var3 ... n]
```

18/06/2022

```
l1 = [1,2,3,4,5]
l1[2] = 10
print(l1)
```

O/P: [1, 2, 10, 4, 5]

```
del l1[0]
print(l1)
```

O/P: [2, 10, 4, 5]

```
del l1[1:3]
print(l1)
```

O/P: [2, 5]

```
del l1[0:2]
print(l1)
```

O/P: []

* `l1=[1,2,3,4,5]*2`
`print(l1)`

O/P: `[1,2,3,4,5] [1,2,3,4,5]`

* `l1=[1,'a',"abc",[2,3,4,5],8.9]`

```
i=0
while i<(len(l1)):
    print(l1[i])
    i+=1
print(l1[3][1])
```

O/P:

```
1
a
abc
[2,3,4,5]
8.9
3
```

⇒ List inside a list is called Nested lists.

* Cloning of lists:

`clone = copy`

→ cloning is to create separate copy of the list. slice operation is used to clone the list.

```
* l1=[1,2,3,4,5,6,7,8,9,10]
  l3=l1[2:6]
  print(l3)
```

O/P:

`[3,4,5,6]`

* Basic List Operations:

1) len(l1)
2) concatenate `l=l1+l2`
3) repetition (*) `l=["Hello","world"]*2`
   `print(l)`.

4) In and Not in:

```
if a in ['a', 'e', 'i', 'o', 'u']:
    print("Yes")
else:
    print("No")
```

(a not in)

5) max:

```
l1 = [1, 3, 9, 6]
print(max(l))
```

6) min: `print(min(l))`

7) sum: `print(sum(l))` : prints sum of all elements in list.

8) all: All the elements in the list must be non-zero : prints True. (or) False. (Boolean value).

9) any: Any one element in the list must be non-zero ; returns Boolean value.

10) list:

```
l = list("hello")
print(l)
```

O/P: ['h', 'e', 'l', 'l', 'o']

11) sorted: returns a sorted list and does not disturb the original list.

```
l2 = sorted(l)
print(l2)
```

O/P: [1, 3, 6, 9]

List Methods:

- Must be called on list objects.

1) append: `list.append(obj)`

Eg:
```
l = [6,3,7]
l.append(10)
print(l)
```
O/P: [6,3,7,10]

* Adds the elements at the end.

2) count: counts the number of times a character is present in the list

⇒ `list.count(obj)`

Eg: `l.count(6)` O/P: 1

3) Index: Returns the index of a given character.

`list.index(obj)`

Eg: `l.index(3)` O/P: 1.

4) Insert: Inserts an object at a specified location

`list.insert(index,obj)`

Eg: `l.insert(3,100)` O/P: [6,3,7,100]

5) pop: returns the element it deleted.

`list.pop([index])`

→ If index is specified it deletes specific object

→ If no index: deletes the last element

6) remove: `list.remove(obj)`

if we do not know the index we can directly specify the object

7) reverse(): `list.reverse()`

Eg: l = [6,3,7]

O/P: [7,3,6]

```
l.reverse()
print(l)
```

8) sort(): `list.sort()`

Eg:
```
l.sort()
print(l)
```

O/P: [3,6,7]

9) extend(): `list.extend(list2)`

Eg:
```
l1 = [1,2,3]
l2 = [4,5,6]
l1.extend(l2)
print(l1)
```

O/P: [1,2,3,4,5,6]

```
l1.append(l2)
print(l1)
```

O/P: [1,2,3,[4,5,6]]

* Insert, remove and sort do not return any object but just modifies the original.

Eg: l1 = ['1','a',"abc",'2','B',"DEF"]

```
l1.sort()
print(l1)
```

l[2:5] = [ ] ⟹ removes the elements at specified index
print(l)

O/P: ['1', '2', 'a', "abc", 'B', "DEF"]

['1', '2', "DEF"]

O/P: ['1', '2', 'B', "DEF", "a", "abc"]

['1', '2', "abc"]

* Data Structure:
  - Stack
  - Queue.

* The element inserted at the last will be deleted out first in a stack.

⟹ LAST IN FIRST OUT [LIFO]

→ push() : insert the elements.
→ pop() : delete the elements.
→ peep() : allows the user to see the last element entered the stack.

To make the list a stack.

push: append.
pop: pop
peep: index.

⟹ FIRST IN FIRST OUT [FIFO] ⟹ for a queue.

The first entered element will be deleted first.

21/06/2022

* Queue (FIFO): for list to change queue.

push() : append()
pop() : pop(0)
peep() : l[-1]

* LIST COMPREHENSIONS:

*) WAP to create a list of cubes of numbers from 1 to 10.

```
l = [ ]
for i in range(11):
    x = i**3
    l.append(x)
print(l)
```

Output:

[1, 8, 27, 64, 125, 216, 343, 512, 729, 10

list [expression for variable in sequence]

* Python supports computed list called list comprehensions. It helps programmers to create list in consist way (It creates new list where each element is obtained by applying some operations to each member of another sequence (or) iterable)

* List Comprehensions can be used to combi

the elements of 2 lists.

Ex: l1=[i**3 for i in range(1,11)]
print(l1)

Ex: print([(x,y) for x in [10,20,20]
for y in [30,10,40] if x!=y])

[(10,30),(10,40),(20,30),(20,10),(20,40),
(30,10),(30,40)]

*Looping in lists:

Ex: l=[1,2,3,4]
for i in l:
print(i)

for i in range(len(l)):
print(l[i])

*Using enumerate() function:

Ex: for index,i in enumerate(l):
print(i,"is present at",index)

O/P: 1 is present at 0
2 " at 1
3 " at 2
4 " at 3

*Using an iterator:

iter() function

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

```
it = iter(l)
for i in rangl(len(l))
    print(next(it))
```

* Functional Programming:

- filter() → (func, sequence)
- map()
- reduce().

* ~~Syntax~~

⇒ Functional programming decomposes a problem into set of functions.

1) filter(): constructs a list from those elements of the list for which a function returns true.

filter(function, sequence)

Ex: WAP to create a list of numbers divisible by 2 (or) 4 using list comprehensions.

```
* def check(x):
        if (x%2==0 or x%4==0):
            return 1

evens = list(filter(check, range(2,22))
print(evens)
```

O/P: [2, 4, 6, 8, 10, 12, 14, 16, 18, 20]

22/06/2022

2) map():

map(function, sequence)

- Map function applies a particular fn to every element of list and returns the modified list.

Ex: WAP to add '2' to every element in the list.

```
* def add(x):
      x+=2
      return x

  l=[1,2,3,4,5]
l1=list(map(add, l))
  print(l)
```

O/P:
[3,4,5,6,7]

Ex: WAP to add the corresponding elements of 2 lists using map function

```
* def addl(x,y):
      return x+y

l1=[1,2,3,4,5]
l2=[10,20,30,40,50]
l=list(map(addl, l1, l2))
print(l)
```

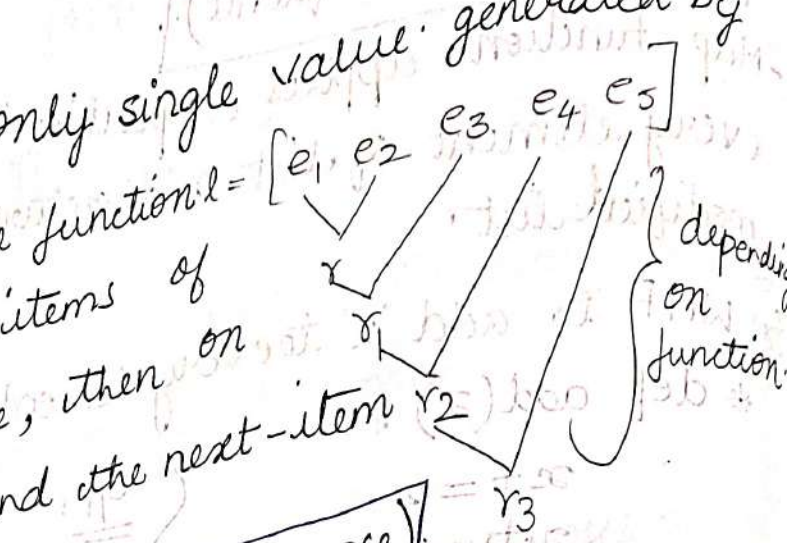
O/P:
[11,22,33,44,55]

```
l1=[1,2,3,4,5]
l2=[10,20]
l=list(map(addl, l1, l2))
```

O/P:
[11,22]

3) reduce():

- reduces the list of elements into a single value.
- returns only single value generated by calling the function on first 2 items of the sequence, then on the result and the next item and so-on.



* reduce (function, sequence)

*Ex: WAP to calculate sum of the values in a list using reduce.

```
* de
* import functools
  def add(x, y):
      return x+y

  l = [1, 2, 3, 4, 5]
  r = functools.reduce(add, l)
  print("Total=", r)
```

O/P:

Total = 15

```
* l = [2]
  r = functools.reduce(add, l)
  print("Total=", r)
```

O/P:

Total = 2

Ex: WAP to split the list of numbers into even & odd number list.

```
* l=[] ; le=[] ; lo=[]
  x=int(input("Enter the no. of elements in list"))
  for i in range(x):
      y=int(input("Enter a number"))
      l.append(y)

  print(l)
  for i in l:
      if (i%2==0):
          le.append(i)
      else:
          lo.append(i)
  print("Even numbers=", le)
  print("Odd numbers=", lo)
```

*WAP to print index at which a particular value exist. If the value exist at multiple locations in the list; print all the indices. Also count the no. of times the value is repeated in the list.

```
import re
* l=[]
  n=int(input("Enter the no. of elements in list"))
  for i in range(n):
      x=int(input("Enter a no"))
      l.append(x)

  print(l)
```

```
c = int(input("Enter the element to be
                checked")).
count = 0
for i in range(n)
    if (l[i]==c):
        print(i)
        count+=1
print("Count = ", count).
```

*WAP to create a list of first 20 odd numbers using short-cut (List comprehensions) method:

~~print([for i in range(20) 2*i+1])~~.

```
print([2*i+1 for i in range(20)])
```

O/P:

[1, 3, 5, 7, 9, 11, 13, 15, 17, 19]

*WAP to find largest value in a reduce function

```
import functools.
l = []
n = int(input("Enter no. of elements"))
for i in range(n):
    x = int(input("Enter a no")).
    l.append(x)
def max(x, y):
    return x>y
```

~~l=list(reduce(max,l))~~

```
l=list(functools.reduce(max,l))
print("largest=",l)
```

* WAP that has a list of 4 positive and 4 negative numbers. Create another list using filter that has only +ve values.

```
l=[]
n=int(input("Enter no.of elements"))
for i in range(n):
    x=int(input("Enter a no"))
    l.append(x)
print(l)
def pos(x):
    if x>=0:
        return x
l2=list(filter(pos,l))
```

* WAP that converts uppercase string into a string of lowercase characters using map functions.

```
l=[]
n=int(input("Enter no.of elements")
for i in range(n):
    x=int(input("Enter a no"))
    l.append(x)
```

```
def convert(x):
    return x.lower()

l = list(map(convert, l))
print(l)
```

* WAP to add 2 matrices:

```
l1 = [[10,20,30],[40,50,60]]
l2 = [[1,2,3],[4,5,6]]
l = [[0,0,0],[0,0,0]]
for i in range(len(l1)):
    for j in range(len(l1[i])):
        r[i][j] = l1[i][j] + l2[i][j]

for k in range(len(r)):
    print(k)
```

24/06/2022

TUPLES: ( ) → Collection of data separated by commas in parantheses.

- Data structure

t1 = (Var1, Var2, Var3 :----)

Ex: t = (1,2,3,4)
print(t)
O/P: (1,2,3,4)

→ t = ("abc", "xyz")
print(t)
O/P: ('abc', 'xyz')

→ t = (1, "abhinav", 'A', 9.9)
print(t)
O/P: (1, 'abhinav', 'A', 9.9)

→ t = ()
print(t)
O/P: ( )

→ a = 10
b = 20
print(a,b)
O/P: 10 20
Represents a Tuple

- Tuples are immutable i.e. cannot be modified.
  - Cannot use insert, sort, delete, reverse.

* A tuple is a sequence of immutable objects.
* To create a tuple with a single element we must add a comma after the element otherwise it is treated as ordinary datatype.

Ex: t = (10,)
print(type(t))

O/P: tuple

* Tuples are useful for representing record which store related information about a subject together.
* Some of the built-in functions return tuple.
* DIVMOD function:

*** divmod() function**

quo, rem = divmod(25,4) ⇒ Returns a tuple
print(quo)
print(rem)

O/P: 6
1

* ACCESSING THE VALUES: (Slice Operation)

T = (1,2,3,4,5,6,7,8,9,10)
print(T[3:6])
print(T[:4])
print(T[4:])
print(T[:])

O/P: (4,5,6)
(1,2,3,4)
(5,6,7,8,9,10)
(1,2,3,4,5,6,7,8,9,10)

Slice Operation returns a tuple: when we pass a tuple.

Ex:
```
t1 = (1,2,3)
t2 = (4,5,6)
t = t1 + t2
print(t)
```
O/P: (1,2,3,4,5,6)

* Can extract values from tuple to form another tuple.

```
* t1 = (1,2,3)
del t1[1]
print(t1)
```
Error.

del t1 ⇒ possible : complete tuple is deleted.

* BASIC TUPLE OPERATIONS:

1) length : len(t) : gives no. of elements.

2) Concatenation : $t_1 + t_2$: combines 2 tuples.

3) Repetition : ('GOOD',)*3

4) membership : 3 in (1,2,3) True
5 not in (1,2,3) False.

5) max : max(t_1)
min : min(t_1).

6) Iteration :
```
for i in (1,2,3):
    print(i," ")
```

7) tuple() :
```
• print(tuple("Hello"))
```
O/P: ('H','e','l','l','o')
```
• print(tuple([1,2,3]))
```
O/P: (1,2,3).

* T1 = (1, 2, 3)
T2 = (1, 2, 3)
print (T1 > T2) → False
print (T1 == T2) → True
print (T1 < T2) → False

Check
gt no. of elements or equal elements

* TUPLE ASSIGNMENT:

* Tuple assignment allows the tuple of variables on the left side of the = operator to be assigned the values from a tuple given on the right side of operator.
* Each value is assigned to its respective variable

* (V1, V2, V3) = (1, 2, 3)
print (V1, V2, V3)
O/P: 1 2 3

* T = (10, 20, 30)
(V1, V2, V3) = T
print (V1, V2, V3)
O/P: 10 20 30

* (V1, V2, V3) = (2+4, 5/3+4, 9%6)
print (V1, V2, V3)
O/P: 6 5.666667 3

6)9 (1
6
3

* (V1, V2, V3) = (1, 2)
print (V1, V2, V3)
Error

* Tuples for returning multiple values:

* WAP to return highest & lowest values in a list.

```
def high_low(l):
    x, y = max(l), min(l)
    return (x, y)

l = [1, 2, 3, 4, 5]
(max_v, min_v) = high_low(l)
print(max_v)
print(min_v)
```

O/P:

```
99
82
```

25/06/2022

* NESTED TUPLES:

Ex: `Toppers = (("Raj", "CSE", 9.2), ("Asha", "IT", 9.1))`

```
for i in toppers:
    print(i)
```

O/P: ('Raj', 'CSE', 9.2)
('Asha', 'IT', 9.1)

→ WAP to print the name of the student and for marks in 3 subjects where marks are specified as a list.

@ → `T = ("XYZ", [15, 25, 30])`

→ INDEX METHOD:

SYNTAX:

```
seq.index(obj)
```

* If the searched element is not in the list (or) tuple; an error is generated.

ex: T = (1,2,3,4,5)
print(T.index(3)) ⇒ 2
print(T.index(9)) ⇒ error.

→ COUNT METHOD:

** Returns a number of ~~times~~ elements with a specified value in a tuple.

ex: t = "abcdabcd"
print(t.count(c)) ⇒ 2.

→ List Comprehensions and Tuples:

ex: def double(T):
    return ([i*2 for i in T])

• T = 1,2,3,4,5.

T = (1,2,3,4,5)
print(double(T))

O/P: [2,4,6,8,10]

→ The operations are performed on the tuple and the results are stored as list elements.

VARIABLE LENGTH ARGUMENTS: TUPLES

```
def func(*T):
    print(T).
T = (40,50,25,1,06,100,10)
fun(T)
```

O/P:
(40,50,25,1,06,100,10)

→ Also known as gather.

→ It allows a function to accept variable number of arguments (*)

```
T= (56,3)
q,r = divmod(T)
print(q,r)
```

} Error.

```
T=(56,3)
q,r=divmod(*T)
print(56,3)
```

O/P: 18 2

* Zip() function:

The zip function takes 2 or more sequences and zips them together into a list of tuples.

The tuple thus formed has one element from each sequence.

```
t=(1,2,3)
l=("akshay", "Aakash", "Pavan")
```

print(list(zip(T, l)))

O/P: [(1, 'Akshay'), (2, 'Aakash'), (3, 'Pavan')]

Ex: T = (1, 2)
l = ("Akshay", "Aakash", "Pavan")
print(list(zip(T, l)))
O/P: [(1, 'Akshay'), (2, 'Aakash')]

→ We can use enumerate function on tuples.

* for i, e in enumerate('ABC'):
    print(i, e)

O/P:
0 A
1 B
2 C

* T = ((1, 'a'), (2, 'b'))
for i, c in T:
    print(i, c)

O/P:
1 'a'
2 'b'

* l = [1, 2, 3]
print(l)
print(tuple(l))

O/P:
[1, 2, 3]
(1, 2, 3)

- Computation of tuples
- Interpreted languages take long time for execution.

* String formatting fundions:

```
T = ("abc", 89)
print("%s got %d" % (T[0], T[1]))
```

- type should match; no exact reorder.

* WAP using a function that returns the area and circumference of a circle whose radius is passed as an argument.

```
def circle(r):
    area = 3.14 * (r**2)
    C = 2 * 3.14 * r
    return (C, area)
x = int(input("Enter the radius"))
(r,s) = circle(r)
print("circumference = ", r)
print("Area = ", s).
print("circumference=%d \n area =%d", c, a).
```

O/P:

```
* t = (1,10,2,25)
  z = sorted(t)
  print(z).
```

O/P: [1,2,10,25]

Sorted; when applied on a tuple: it returns list.

→ Sorted function takes any sequence and returns a new list with same elements but in different order

* Format specifiers cannot be used on list

* **SETS** : - mutable
- data structure

Strings, Lists, Tuple } ordered sequence.

- Sets are unordered sequence.
→ List can have duplicates
→ Sets cannot have duplicates.

* s = {1, 2, 3, 1, 2}.
print(s)

O/P: {1, 3, 2}
No error.

- Can store different types of data.
- Enclosed between { }

- S = set() : creates an emptyset.

⇒ SYNTAX:

S_var = {V1, V2, V3, ...}

* Ex:
s = {1, 2.0, "abc"} { O/P: {1, 2.0, 'abc'}.
print(s)
s = set([1, 2.0, 'abc']) { O/P: {1, 2.0, 'abc'}
print(s)
LI = [1, 2, 3, 4, 5, 6, 4, 3, 2, 1] { O/P: {1, 2, 3, 4, 5, 6}
print(set(LI))
str = "abcdefgabcdefg" { O/P: {'a', 'b', 'c', 'd', 'e', 'f', 'g'}
print(set(str))
TI = ('a', 'b', 'c', 'd', 'b', 'e', 'a') {'a', 'b', 'c', 'd', 'e'}.
print((TI))
print(set("She sells sea shells on sea shore".split()))

O/P: {'She', 'sells', 'sea', 'on', 'shore', 'shells'}

* S1 = {1, 2, 3, 4}
S2 = {1, 3, 5, 6}
S1 ∪ S2 = {1, 2, 3, 4, 5, 6}
S1 ∩ S2 = {1, 3}.
S1 - S2 = {2}.

S1.union(S2) : modifies S1
S1.intersection(S2) : creates a newset

S1.symmetric_difference(S2) : returns a [S1^S2] set containing all the elements

S1.union(S2) [S1|S2]
S1.intersection(S2) [S1&S2]
S1.difference(S2) [S1-S2]
S1.symmetric_difference(S2)
S1.intersection_update(S2)
S1.difference_update(S2)
S.issuperset(t) [S>=t]
S.issubset(t) [S<=t]

except the common elements.

* s={1,2,3}
  t={4,5,6}
  s.update(t)
  print(s)
  {1,4,2,5,3,6}.

s.add(x) : adds a single element to the set.

s.add(10)
print(s)
O/P {1,2,3,10}

s.update(t) : adds a set to the given set.

→ Update method can take tuples, list, strings or other sets as its arguments.

Ex: s={1,2,3}
t={4,5,6}

s.remove(4) → Error
s.remove(3) → s={1,2}
s.discard(4) → No error if object is not there.
s.pop() → removes an arbitrary element (Random)

s.clear → remove all the elements.
print(s.clear) → set()
↳ do not give {}.

* len(s) ⇒ returns no. of elements in the list.
* in, not-in ⇒ checks for a object the set.

t=s.copy() ⇒ All elements refers to the first set elements.

* disjoint

isdisjoint() ⇒ if 2 sets have elements common it is true else false.

all(): returns True if all elements are non-zero

any():

max(): maximum element in set

min(): minimum element.

sum(): adds all non-duplicate Elements.

* s={1,2,3}

t={3,2,1}

s==t → True.

* A set cannot contain other mutable objects.

s={10,20,[30,40]} ✗

* Since sets are unordered, indexing have no meaning.

Set operations does not allow users to access or change an element using indexing or slicing.

*WAP to iterate through a set:

```
s = set("Hello All, Good Morning")
for i in s:
    print(i, end=" ")
```

O/P: {'H', 'e', 'l', 'o', 'A', 'G', 'M', 'd', 'r', 'n', 'i', 'g'}

28/06/2022

* Dictionary: → not a sequence; it is a mapp
→ unordered

- data structures in python

(key-value pairs are stored)

↓

should be unique.

- enclosed in flower brackets.

* dict = { } → empty dictionary.

* SYNTAX: key and value are string types.

dict_var = {key1: value1, key2: value2, ...}

- All the keys should not be mutable.

key → number
→ string
→ Tuple.

} unique.

Values can be any type of data.

stud = {'RNo': '733-001', 'Name': 'Abhi',
'Course': 'BE'}

len(stud) ⟹ Output: 3

print(stud) ⟹ prints the complete dictionary

o/p: {'Name': 'Abhi', 'RNo': '733-001', 'course': BE}

print(stud[key]) ⟹ To return individual element from a dictionary. (also called hashing)

↳ prints corresponding value

if the given key is not available in the dictionary; we will get a key error.

* stud = {'RNo': '733-001', 'name': 'Abhi', 'course': 'BE', 'RNo': '733-011'}

* stud['course'] = 'M.Tech'

↳ modifies the course value.

↳ no error but modifies the value of the key

*Delete:

* del stud['Name']
print(stud)

o/p: {'RNo': '733-011', 'course': 'M.Tech'}

→ stud.clear()
print(stud)

o/p: { }.

→ del stud
print(stud)

o/p: error

↳ removed completely.

→ Dictionary is a data structure in which we store key-value pairs.

→ Dictionaries are mappings (collection of objects that store objects by key instead of relative position (index)).

→ Using a mutable object as key gives a type-error.

* We can create a dictionary using dict() function.

Ex: print(dict([['Rollno', '465'], ('course', 'PP')]))

tuple is considered as key-value pair

O/P: {'Rollno': '465', 'course': 'PP'}

* Dictionary Comprehensions:

Syntax:

dic = {exp for var in seq [if cond]}

⇒ WAP to create 10 key value pairs where key is a no. in the range 1 to 10 and value is twice the number.

```
dic={x:2x for x in range(1,11)}
print(dic)
```

O/P: {1:2, 2:4, 3:6, 4:8, 5:10, 6:12, 7:14, 8:16, 9:18, 10:20}

* d= {'RollnO': 65, 'course': 'PP'}

d.pop() → deletes any random key-value pair.

d.pop('RollnO') → deletes the specific key-value pair and returns its value

⇒ Syntax:

dict.pop(key, [default])

* d.pop('marks', -1) ⇒ returns -1 instead of showing an error. otherwise returns the respective value.

* dict.popitem() ⇒ deletes a random key-value pair; we cannot specify a key.

* Ex: d= {'RNo': '002', 'Name': 'Abhi', 'course': 'BE'}

```
if 'course' in d:
    print(d['course'])
```

O/P:
BE

print(sorted(d.key())) → sorts all the keys

o/p ['course', 'name', Rnb']

Looping over a dictionary:

- Keys : only keys are returned
- Values: only values are returned
- Items: both key & value are returned.

```
d={'RNo':'052', 'Name':'Sivani', 'Course':
for i in d.values():
    print(i)
for i in d.keys():
    print(i,d[i])
for i,j in d.items():
    print(i,j)
```

* Value can be a dictionary.

* Nested Dictionaries:

```
students={'Raj':{'PP':90,'BEE':92},
          'Ram':{'PP':99,'BEE':80}}
for key,value in students.items():
    print(key,value)
```

→ Sorted function takes any sequence and returns a new list with same elements but in different order

* Format specifiers cannot be used on list

* SETS : - mutable
- data structure

Strings, lists, tuple } ordered sequence

- Sets are unordered sequence
→ List can have duplicates
→ Sets cannot have duplicates

```
* s={1,2,3,1,2}
  print(s)
```
O/P: {1,3,2}
No errors

- Can store different types of data
- Enclosed between { }
- S=set() : creates an emptyset

⇒ SYNTAX:

```
S_var1 = {v1, v2, v3, ....}
```

* Ex:

```
→ s={1,2.0,"abc"}
  print(s)
```
O/P: {1, 2.0, 'abc'}

```
→ s=set([1,2.0,'abc'])
  print(s)
```
O/P: {1, 2.0, 'abc'}

```
→ L1=[1,2,3,4,5,6,4,3,2,1]
  print(set(L1))
```
O/P: {1,2,3,4,5,6}

```
→ str="abcdefgabcdefg"
  print(set(str))
```
O/P: {'a','b','c','d','e','f','g'}

```
→ T1=('a','b','c','d','b','e','a')
  print((T1))
```
{'a','b','c','d','e'}

```
→ print(set("She sells sea shells on sea shore".split()))
```
O/P {'She', 'sells', 'sea', 'on', 'shore', 'shells'}

```
* S1={1,2,3,4}
  S2={1,3,5,6}
  S1 U S2 = {1,2,3,4,5,6}
  S1 ∩ S2 = {1,3}
  S1-S2 = {2}
```

S1.union(S2) : modifies S1
S1.intersection(S2) : creates a newset
S1.symmetric_difference(S2) : returns a set containing all the elements [S1^S2]

S1.union(S2) [S1|S2]
S1.intersection(S2) [S1&S2]
S1.difference(S2) [S1-S2]
S1.symmetric_difference(S2)
S1.intersection_update(S2)
S1.difference_update(S2)
S.issuperset(t) [S>=t]
S.issubset(t) [S<=t]

except the common elements.

* s={1,2,3}
  t={4,5,6}
  s.update(t)
  print(s)
  {1,4,2,5,3,6}.

s.add(x) : adds a single element to the set.

s.add(10)
print(s)
} O/P: {1,2,3,10}

s.update(t) : adds a set to the given set.

→ Update method can take tuples, list, strings or other sets as its arguments.

Ex: s={1,2,3}
t={4,5,6}

s.remove(4) → Error
s.remove(3) → s={1,2}
s.discard(4) → No error if object is not there.
s.pop() → removes an arbitrary element (Random)

s.clear → remove all the elements.
print(s.clear) → set()
↳ do not give {}.

* len(s) ⇒ returns no. of elements in the list.
* in, not in ⇒ checks for a object the set.

t=s.copy() ⇒ All elements refers to the first set elements.

* disjoint
isdisjoint() ⇒ if 2 sets have elements common it is true else false.

all() : returns True if all elements are non-zero
any() :
max() : maximum element in set
min() : minimum element.
sum() : adds all non-duplicate Elements.

* s={1,2,3}
  t={3,2,1}
  s==t → True.

* A set cannot contain other mutable objects.
s={10,20,[30,40]} ✗

* Since sets are unordered, indexing have no meaning.
Set operations does not allow users to access or change an element using indexing or slicing.

* WAP to iterate through a set:

```
s = set("Hello All, Good Morning")
for i in s:
    print(i, end=" ")
```

O/P: {'H', 'e', 'l', 'o', 'A', 'G', 'M', 'd', 'r', 'n', ' ', 'i', 'g'}

28/06/2022

* Dictionary: → not a sequence; it is a mapping
  → unordered
- data structures in python
(key-value pairs are stored)
↓
should be unique.
-enclosed in flower brackets.

* dict = { } → empty dictionary.

* SYNTAX: key and value are string types.

dict_var = {key1: value1, key2: value2, ...}

• All the keys should not be mutable.

key → number, string, Tuple. } unique.

Values can be any type of data.

Ex:
stud = {'RNo': '733-001', 'Name': 'Abhi', 'Course': 'BE'}

len(stud) ⟹ Output: 3

print(stud) ⇒ prints the complete dictionary

O/P: {'Name': 'Abhi', 'RNo': '733-001', 'Course': 'BE'}

print(stud[key]) ⇒ To return individual element from a dictionary (also called hashing)
↳ prints corresponding value

if the given key is not available in the dictionary; we will get a key error.

* stud = {'RNo': '733-001', 'name': 'Abhi', 'Course': 'BE', 'RNo': '733-011'}

* stud['course'] = 'M.Tech'
↳ modifies the course value
↳ no error but modifies the value of the key

* Delete

* del stud['Name']
print(stud)
O/P: {'RNo': '733-011', 'course': 'M.Tech'}

→ stud.clear()
print(stud)
O/P: { }

→ del stud
print(stud)
O/P: error
↳ removed completely.

→ Dictionary is a data structure in which we store key-value pairs.

→ Dictionaries are mappings (collection of objects that store objects by key instead of relative position (index)).

→ Using a mutable object as key gives a type-error.

* We can create a dictionary using dict() function.

Ex: print(dict([['Rollno', '165'], ('course', 'PP')]))

tuple is considered as key-value pair

O/P: {'Rollno': '165', 'course': 'PP'}

*Dictionary Comprehensions:

Syntax:

dic = {exp for var in seq [if cond]}

⇒ WAP to create 10 key value pairs where key is a no. in the range 1 to 10 and value is twice the number.

dic = {x: 2x for x in range(1,10)}
print(dic)

O/P: {1:2, 2:4, 3:6, 4:8, 5:10, 6:12, 7:14, 8:16, 9:18, 10:20}

* d = {'Rollno': '65', 'course': 'PP'}

d.pop() → deletes any random key-value pair

d.pop('Rollno') → deletes the specific key-value pair and returns its value

⇒ Syntax:

dict.pop(key, [default])

* d.pop('marks', -1) ⇒ returns -1 instead of showing an error. Otherwise returns the respective value.

* dict.popitem() ⇒ deletes a random key-value pair; we cannot specify a key.

* Ex: d = {'RNo': '002', 'Name': 'Abhi', 'course': 'BE'}
if 'course' in d:
print(d['course'])

O/P:
BE

• print(sorted(d.key())) → sorts all the keys.

O/P: ['course', 'name', 'Rno']

* Looping over a dictionary:

- Keys : only keys are returned
- Values: only values are returned
- Items: both key & value are returned.

```
d={'RNo':'052', 'Name':'Sivani', 'Course':'PP'}
for i in d.values():
    print(i)
for i in d.keys():
    print(i,d[i])
for i,j in d.items():
    print(i,j)
```

* Value can be a dictionary.

* Nested Dictionaries:

```
students={'Raj':{'PP':90, 'BEE':92},
          'Ram':{'pp':99, 'BEE':80}}
for key,value in students.items():
    print(key,value)
```

Check:

```
for i in students():
    print(i)
```
} Returns keys

02/07/2022

* To get a value we need to specify the key.

⇒ d[key]

Ex:

```
stud={1:{'RNo':'001', 'Name':'Ram'},
      2:{'RNo':'010', 'Name':'Sivani'}}
print(stud[1])
```

O/P:
{'RNo':'001', 'Name':'Ram'}

⇒ We can add a new key-value pair to the dictionary and also we can update an existing key-value pair

```
* del stud[1]['Name']
print(stud[1])
stud[1]['Name']='Bhupathi'
print(stud[1])
stud[4]={}
stud[4]['rno']='65'
del stud[1],stud[2]
```

O/P:
{'RNo':'001', 'marks':98}

O/P:
{'RNo':'001', 'marks':98, 'Name':'Bhupathi'}

⇒ Both key-value pairs can be deleted

* ID = {1,2,3,4}

student_details = ({'Name': 'XYZ', 'Branch': 'CSE'})

d = dict(zip(ID, student_details))

O/P: {1: {'Name': 'XYZ', 'Branch': 'CSE'}}

*Built-in Dictionary Functions and Methods:

- len(d)
- str(dict) ⇒ represents the dictionary in the form of string
- d.clear()
- d.copy() = d1
- d.get(key)
- d.has_key(key)
- d.items()
- d.keys()

Check:
⇒ stud[10] — key error

d.get(10) ⇒ returns None if key is not present in the dictionary.

stud.has_key(2) → returns true if present in the dictionary and false if it is not in the dictionary.

- d.setdefault(key, value)
- d.update(d1) → add key value pair of d1 to d.
- d.values()
- d.iteritems()
- k in d
- k not in d

NumPy:

Numerical Python

ndarray: array

import numpy

Pip install numpy

Check:

```
for i,j in d.iteritems():
    print(i,j)
```

Imp: *

05/07/2022

*a = {1:2, 3:4}

```
b = a
print(b)
b[3] = 10
print(str(a))
print(b)
```

O/P:
{1:2, 3:4}
{1:2, 3:10}
{1:2, 3:10}

* a = {1:2, 3:4}

```
b = a.copy()
print(b)
b[3] = 10
print(str(a))
print(b)
```

O/P:
{1:2, 3:4}
{1:2, 3:4}
{1:2, 3:10}

copy: creates an extra memory location.

* Anagram
* difference b/w for and while
* break & continue
* Strings functions (inbuilt)
* in and not in operators

* Slicing Operation
* Size & methods for all the structures
* arguments types
* Sets, lists, tuples
* Regular expression
* % a b i %
* filter, map, reduce
* list, set, tuples differences
* Variations: perfect, armstrong, etc.
* Recursion: Towers of Hanoi etc.

06/07/2022

* NumPy: NUMERICAL PYTHON:

Eg:
```
import numpy
ar = numpy.array([1,2,3,4])
print(ar)
```
O/P: [1 2 3 4]

*
```
import numpy as np
a=np.array((10,20,30))
print(a)
```
O/P: [10 20 30]

*
```
import numpy
ar = numpy.array([1,2,3,4], admin=2)
print(ar)
```
O/P: [[1 2 3 4]]

*
```
import numpy as np
a=np.array([[1,2],[3,4]])
print(a)
print(a.shape)
```
O/P: [[1,2]
[3,4]]
(2,2)

* "ndarray.shape" is an attribute that returns a tuple of array dimensions. It can also be used to resize the array.

* import numpy as np
a = np.array([[1,2,3],[4,5,6]])
print(a.shape)
print(a)
a.shape = (3,2) → original array is modified
print(a)

O/P: (2,3)

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

b = a.reshape(3,2) → new array is created
print(b)
print(a)

O/P: [[1 2]
 [3 4]
 [5 6]]

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

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

* import numpy as np
a = np.arange(20) → default 1D
print(a)
b = a.reshape(2,10)
print(b)

O/P:
[0 1 2 3 4 5 6 7 8 ... 19]

[[0 1 2 3 4 5 6 7 8 9]
 [10 11 12 13 14 15 16 17 18 19]]

* import numpy as np
a = np.zeroes((3,3)) → default 1D
print(a)
b = np.ones((3,3))
print(b)

O/P:
[[0 0 0]
 [0 0 0]
 [0 0 0]]

[[1 1 1]
 [1 1 1]
 [1 1 1]]

* import numpy as np
a = np.array([1,2,3], dtype=float)
print(a)

O/P: [1. 2. 3.]

* datatypes
strings
integers
float
complex
boolean

* Array Indexing:

import numpy as np
a = np.array([10,20,30,40])
print(a[0])
print(a[1]+a[2])

O/P:
10
50

⇒ arr = np.array([[1,2,3,4],[5,6,7,8]])
print(arr[1,3])

O/P: 8

row index → 1
column index → 3

* import numpy as np
a = np.array([[[1,2,3],[4,5,6]],[[7,8,9],[10,11,12]]])
print(a)
print(a.shape)
print(a[0,1,2])

O/P:
[[[1 2 3]
  [4 5 6]]
 [[7 8 9]
  [10 11 12]]]

* Numpy Array slicing:

a = np.array([1,2,3,4,5,6,7])
print(a[1:5])
print(a[-3:-1])
print(a[1:5:2])
print(a[::2])

O/P:
[2 3 4 5]
[5 6]
[2 4]
[1 3 5 7]

* a = np.array([[1,2,3,4,5],[6,7,8,9,10]])
print(a[1,-1])
~~print(a[1:-1])~~

O/P: 10

print(arr[1,1:4]) O/P: [7 8 9]
print(arr[0:2,2]) [3 8]
print(arr[0:2, 1:4] [[2 3 4]
[7 8 9]]

* itemsize:
The attribute numpy.itemsize returns the length of each element of array in bytes.

26/07/2022

* ndim: → gives the dimension of the given array.
ex: a = numpy.array([[1,2,3],[4,5,6]])
↳ 2dim.

* → While using reshape method; if the no. of elements in the new array is more than the previous array; it gives an error.

* asarray:
l = [1,2,3]
a = np.asarray(l)
type(a)
O/P: <class ndarray>

* The data in an array must be of same datatypes or of homogeneous datatype.

* Ex: ~~np.array~~
a = np.array([[10,20,30],[40,50,60]], dtype=float)
print(a.ndim) O/P: 2
print(a.shape) (2,3)
print(a.size) 6
print(a.dtype) float
print(a.itemsize) 8

→ We can create empty array using the method empty.
SYNTAX: np.empty(shape, dtype=float, order='c'/'f')
(mandatory) (default)
↓ row major ↓ column major (default)

ex:
a = np.empty((3,2), dtype=int)
print(a)
(3) → speaks about 1D array of 3 elements.
↳ prints a 3×2 array with some garbage values.

* np.zeroes(shape, dtype= , order)
↳ prints a 3×2 array with zeroes
ones → array with ones

* np.full((2,3), fill_value=8)
O/P: [[8 8 8]
[8 8 8]]
↳ array with all the values as 8

* np.identity(3, dtype=int)
↓ only square matrix
↓ → ∴ no. of rows = no. of columns

o/p:
[[1 0 0]
 [0 1 0]
 [0 0 1]]

* l = [[1,2,3,4,5],[6,7]]
t = np.asarray(l)
print(t)

o/p:
[list[1 2 3 4 5], list[6 7]]

⇒ SYNTAX:

np.arange(start, stop, step, dtype)

ex:
a = np.arange(0, 10, 2, float)
print(a)
o/p: [0. 2. 4. 6. 8.]

* np.linspace(start, stop, count of values required, endpoint, dtype)

eg: a = np.linspace(10, 20, 5)
print(a)
o/p: [10. 12.5 15. 17.5 20.]

if endpoint = False → not considered
→ divides the given range equally into specified parts

* a = np.array(['a', 10.5, 99], dtype=int)
o/p: ⇒ error

b = a.astype(float64)

l1 * l2 → error

* Basic Operations:
* Arithmatic Operators:

27/07/2022

→ a = np.array([[1,2],[3,4]])
print(a+1)
print(a-2)

o/p:
[[2 3]
 [4 5]]
[[-1 0]
 [1 2]]

*operations are done element by element.
* The original matrix is not modified but a new matrix is returned.

print(a*5)
[[5 10]
 [15 20]]

print(a**2)
[[1 4]
 [9 16]]

* a1 = np.array([[1,2],[3,4]])
a2 = np.array([[1,1],[2,2]])
print(a1+a2); print(a1*a2) → array multiplication

⇒ dot
  matmul } ⇒ Methods for matrix multiplication.

a1.dot(a2)
matmul(a1, a2)

* UNARY OPERATORS: a1 = np.array([[1,2],[3,4]])

→ print(a1.max()) = 4

~~print(max(axis~~

print(a1.max(axis=1)) [2 4]

print(a1.min(axis=0)) [1 3]

print(a1.sum()) 10

print(a1.cumsum(axis=1)) [[1 3] [2 6]]

axis=1 → row wise max value/min value

axis=0 → column wise max value/min value.

cumsum → cumulative sum in a row.

*Universal Functions (elementwise)

a = np.array([0,1,2,3])

np.sqrt(a)
np.sin(a)
np.cos(a)
} returns a new array.

np.pi → constant

*Sort arrays:

np.sort(a)

np.sort(a, axis=1)

np.sort(a, axis=0, kind='mergesort')

→ Transpose of a matrix:

a = np.array([[1,2],[3,4]])

print(a.transpose())
print(a.T)
} Transpose.

a = np.array([[0,1,2,3], [4,5,6,7], [8,9,10,11], [12,13,14,15]])

O/P:
[[5 6]
 [9 10]]

print(a[1:3, 1:3])

print(a[:, 3:])

print(np.multiply(M1, M2))

print(np.add(M1, M2))

print(np.subtract(M1, M2))

O/P:
[[3]
 [7]
 [11]
 [15]]

* UNARY OPERATORS: a1 = np.array([[1,2],[3,4]])

→ print(a1.max()) = 4

→ ~~print(max(axis~~

print(a1.max(axis=1)) [2 4]

print(a1.min(axis=0)) [1 3]

print(a1.sum()) 10

print(a1.cumsum(axis=1)) [[1 3] [2 6]]

axis=1 → row wise max value/min value

axis=0 → column wise max value/min value

cumsum → cumulative sum in a row.

*Universal Functions: (elementwise)

a = np.array([0,1,2,3])

np.sqrt(a)
np.sin(a)
np.cos(a)
} returns a new array.

np.pi → constant

*Sort arrays:

np.sort(a)

np.sort(a, axis=1)

np.sort(a, axis=0, kind='mergesort')

→ Transpose of a matrix:

a = np.array([[1,2],[3,4]])

print(a.transpose())
print(a.T)
} Transpose.

* a = np.array([[0,1,2,3], [4,5,6,7], [8,9,10,11], [12,13,14,15]])

O/P: [[5 6] [9 10]]

print(a[1:3, 1:3])

print(a[:, 3:]) [[3] [7] [11] [15]]

print(np.multiply(M1, M2))

print(np.add(M1, M2))

print(np.subtract(M1, M2))

29/07/2022

→ import numpy as np

arr = np.array([1,2,3,4])

x = arr.copy() → creates a copy of the original array.

copy → owns the data

view → do not own the data

arr[0] = 10

print(arr) → [10 2 3 4]

print(x) → [1 2 3 4]

y = arr.view() → only views the array but do not modify

arr[0] = 10

print(arr) → [10 2 3 4]

print(y) → [10 2 3 4]

print(x.base) → if the array owns its data NONE is returned

print(y.base) → otherwise gives the view.

* Iterate Arrays:

```
import numpy as np
arr = np.array([[1,2,3], [10,20,30]])
for x in arr:
    print(x)
```

O/P:
```
[1 2 3]
[10 20 30]
```

```
for x in arr:
    for i in x:
        print(i)
```

O/P:
```
1
2
3
10
20
30
```

for a 3D array: the first element is a 2D array.

As the dimension of an array increases the no. of loops also increases.

```
for x in np.nditer(arr):
    print(x)
```

29/07/2022

FILES: UNIT: 4

Types of files:
- Text file → characters are stored
- Binary file → images; sharing data over internet

→ To open a file:

```
open("filename" [, mode])
```

→ not mandatory to specify

ex: `f = open("data.txt")`

⇓ present in same directory

↳ default mode = read mode.

* Modes

Text	Binary
r	rb
w	wb
a	ab
r+	rb+
w+	wb+
a+	ab+

r+ → read & write
w+ → write & read
a+ → append & read.

* open method in r mode checks if the file exists or not in the directory.
- If file is not in the directory; it gives an error.

w mode: if file is existing; the original data is erased and writes from the beginning.
- if file is not existing it creates a new file.

`f = open("data.txt", "x")`

↓ creates & opens a file.

- If the file is already existing: it gives an error.

t → opens a text file in read mode
b → opens a binary file in text mode.

* read()
  readline()
  readlines()

Ex: f=open("data.txt","r")
c=f.read() → reads all the contents
print(c)

Content in file:
Vasavi
College of
Engineering

O/P:
Vasavi \n College of \n Engineering

c=f.read(6) → first 6 characters are read.
print(c)

O/P: Vasavi

c=f.readline() → read the content of one line.
print(c)
c=f.readline()
print(c)
c=f.readline()
print(c)

O/P: Vasavi
→ Prints a newline
College of.
to indicate newline character

c=f.readlines() → reads all the lines and returns as a list of strings
print(c)

print(f) → prints the filename, mode & address.

f.close() → closes the file.
print(f.closed) → T: if closed
F: if not closed.
print(f.mode) → gives the mode of the file.
print(f.name) → gives the name of the file.

* write()
  writelines()

print(type(f))

f=open("output.dat","w")
l=["\n","\n","\n"]
f.write("Vasavi College")
f.writelines(l). → prints all the strings in the list.

30/07/2022

* Filename: can be given in 2 different paths:
- relative path : Simple filename taken from same directory
- absolute path : complete address of the given file.

* Reading content from a binary file:

Ex:
```
l=[1,2,3,4]
b=bytearray(l)
f=open("file.dat","wb")
f.write(b)
```

→ We cannot move the file pointer

For Binary files:
f.tell() → Current file pointer location
f.seek(3,0/1/2)
↓
no. of bytes to move

0 → beginning

1 → current

2 → end

* Opening file with 'with' keyword:

```
with open("data.txt", "w") as f:
    f.write("Vasavi College")
    print(f.closed)
print(f.closed)
```

} when the pointer comes out the block file is automatically closed

```
with open("data.txt","a") as f:
    f.write("of engineering")
    print(f.closed)
```

```
with open("data.txt","r") as f:
    print(f.read())
    print(f.closed)
    print(f.tell())
    f.seek(0)
    f.seek(3,0)
    print(f.read(20))
```

```
with open("data.txt","r") as f:
    l = f.readline()
    words = l.split('0')
```

* import os

```
os.rename(oldfilename, newfilename)
os.rename("data.txt",'new.txt')
os.remove("new.txt")
os.remove(filename)
```

* Directory Methods:

```
mkdir()
rmdir()
```

```
print(os.getcwd())
os.mkdir("dir")
os.chdir("dir")
print(os.getcwd())
f=open("new2.txt","w")
f.write("hello")
f.close()
os.rm
```

```
import shutil
shutil.rmtree("dir")
```

* Methods from the os module:

→ os.path.abspath(f)
→ os.path.isabs(path)
→ os.path.relpath(path, start)
→ os.path.dirname(path)
→ os.path.basename(path)
→ os.path.split(path)
→ os.path.exists(path)
→ os.path.getsize(path)
→ os.listdir(path)
→ os.path.istile(path)

→ os.path.isdir(path)

02/08/2022

* t = (1,2,3)
  f = open("new.txt", "w") ∴ (1,2,3) → string
  f.write(str(t))
  f.close()
  f = open("new.txt")
  print(f.read())
  x = f.read()
  print(x[1])
  print(x)

O/P:
1
('(', '1', ',', '2', ',', '3', ')')

* t = [1,2,3]

* PICKLING: Process of serialisation and deserialisation.

SERIALISATION: data is converted to byte.

DESERIALISATION: Byte is converted to data.

→ To work with pickling: we need to import the PICKLE package.

Methods:

pickle.dump(d, f1) → dump data 'd' to file pointer f1
x = load(f1)
f1 → file pointer
file should be a binary

* import pickle
  t = (1,2,3,4,5)
  f = open("new.txt", "wb")
  pickle.dump(t,f)
  f.close()
  f = open("new.txt", "rb")
  print(pickle.load(f))
  x = pickle.load(f)
  print(x, type(x))
  print(x[1])
  f.close()

O/P:
(1,2,3,4,5) <class 'tuple'>
2

strip: Removes the extra spaces

d = {0: 'Abhinav', 11: 'Bhupathi', 45: 'Shashank'}
f = open("new.txt", "wb")
pickle.dump(d,f)
f.close()
f = open("new.txt", "rb"); x = pickle.load(f)
print(pickle.load(f), type(x))

* f = open("new.txt")
  x = f.readlines()
  f.close()
  for i in x:
      print(i.rstrip())

Vasavi College of Engineering
CSE Department
CSE

* Write a program to copy the contents of one file to another file.

```
n = input("Enter the file name to be opened:")
f = open(n, "r")
x = f.read()
f.close()

p = input("Enter the file to which data should be copied:")
m = open(p, "w")
m.write(x)
m.close()
f1 = open(m)
print(f1.read())
f1.close()
f2 = open(p, "a")
s = "Sivani"
f2.write(s)
f2.close()
```

* Python pickle module is used for serialising and deserialising python object structures. The process to convert any python objects (list, tuple, dictionary) into byte streams is called pickling (or) flattening / marshalling

We can convert the byte stream back into the python objects by the process of unpickling or deserialisation.

→ Pickling and unpickling are used to easily transfer data from one server to another and store it in a file (or) a database.

* Write a program to count tabs, spaces, newline character.

```
t=0; s=0; n=0
f = open("input.txt", "r")
x = f.read()
f.close()
for i in x:
    if i == '\t':
        t += 1
    elif i == '\n':
        n += 1
    elif i == ' ':
        s += 1
    else:
        pass
print("No. of tabs:", t \n "No. of spaces:",
      s \n "No. of newline characters:",
      n)
```

* WAP that fetches data from a specified URL and print it on the screen.

```
import urllib.request
x = urllib.request.urlopen("https://www.vce.ac.in")
print(x.read())
```

→ PYTHON JSON:

↳ Java script objects.

- Json is a text written with java script object notation which is used for storing & exchange data.
- Python has a built-in package called json which is used to work with json data
- We use json.dumps method to convert of python to json objects.
- We use json.loads method to convert json to python

* Equivalence of python objects to json:

Python	Json
dict	Object
list	Array
tuple	Array
str	string.
int	Number
float	Number
True	true
False	false
None	null

Starts with capital letter

```
import json
x=json.dumps({1:'Ram', 2:'Sita'})
print(x)                          → {1: Ram, 2: Sita}
y=json.dumps([1,2,3,4,5])
print(y)                          → [1, 2, 3, 4, 5]
z=json.dumps((1,2,3,4,5))
print(z)                          → [1, 2, 3, 4, 5]
s=json.dumps("Sivani")
print(s)                          → "Sivani"
p=json.dumps(100)
print(p)                          → 100
a=json.dumps(7.3)
print(a)                          →
t=json.dumps(True)
print(t)                          → true
u=json.dumps(False)
print(u)                          → false
v=json.dumps(None)
print(v)                          → null
```

```
*import json
x={"name": "john", "age": 30, "married": True,
   "divorced": False, "children": ("Ann", "Billy"),
   "pets": None,
   "cars": [{"model1": "BMW", "model2": "Ford Edge"}]
}
```