```
        print(sey.exp).
t1 = Teacher("xyz", 35, 13)
t1.displaydata()
```

→ Method overidding is the ability of a class to change the implementation of a method provided by one of its ancestors.

* MULTIPLE INHERITANCE:

In multiple inheritance a class can be derived from more than one base class.

* Syntax:

```
class base1:
        statement block
class base2:
        statement block
class derived(base1, base2):
        statement block
```

Example:

```
class Base1(object):
        def __init__(sey):
                print("Base1 class")
                • super(base1,sey).__init__()
class Base2(object):
        def __init__(self):
                print("Base2 class").
class Derived(base1, base2):
        super(Derived,sey).__init__
        pass

D=derived()
```

O/P:

Base1 class.
Base2 class.

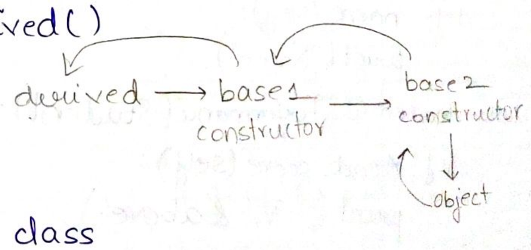
*Acc. MRO: only base1 constructor is executed.

23/08/2022

```
class Base1(object):
        def a __init__(sey):
                super(Base1,sey).__init__()
                print("Base1 class").
class Base2(object):
        def __init__(sey):
                super(Base2,self).__init__()
                print(" Base2 class)
class Derived (Base1,Base2):
        def __init__(sey):
                super(Derived,sey).__init__()
                print("Derived class)
D = Derived()
```

If only method
derived
↓
base1
↓
base2

If constructor
derived
↓
base1
↓
base2
super()



O/P:

Base 2 class
Base 1 class
Derived class.

SIVANI PYTHON PROGRAMMING

* Multi-level Inheritance:

```
*>
class person:
      def name(sey):
            print("Name:")
class Teacher(Person):
      def Qualification(sey):
            print ("Qualification:PhD")
class   HOD (Teacher):
      def experience (sey):
            print (" Atleast 15years:")

hod = HOD( )
hod.name( )
hod.Qualifications( )
hod. experience( )
```

O/P:
Name
Qualification
Atleast 15years

```
*-> class Student:
        def name (sey):
            print("Name:")
    class Academic_Performance(Student):
        def Acad_score (sey):
            print ("90%. & above")
    class ECA(Student):
        def ECA_score(sey):
            print("ECA score: 60%. & above")
class Result (Academic-performance ,ECA):
      def Eligibility(sey):
            print(" Minimum Eligibility")
            sey.Acad_score( )
            sey.ECA_score( )

R= Result ( )
R. Eligibility ( )
```

O/P
Minimum Eligibility
Score - 90%. & above
ECA score - 60%. & above

* OPERATOR OVERLOADING: (Polymorphism)

*-> A single operator can act in different ways in different contexts

-> Operator overloading allows the programmers to extend the meaning of existing operators so that in addition to the basic datatypes, they can also be applied to user-defined datatypes.

```
* class complex :
      def __init__(sey):
            sey.real = 0
            sey.img = 0
      def set value (sey, real, img):
            sey.real = real
            sey.img~~f~~ = img

      def display(sey):
            print(" (", sey.real, "+", sey.img, "i")
      def __add__(sey, C):
            temp = complex()
```

```
c1 = complex( )            → Temp. real = sely.real + c.real
c1.setvalue (1, 2)           Temp.img = sely.imgt c.img.
c2.complex( )                return temp
c2.set value(3,4)
c3 = complex()
c3 = c1+c2
c3.display()
```

27/08/22

*OPERATORS & THEIR CORRESPONDING FUNCTION NAMES:

+:	__add__
+=:	__iadd__
-:	__sub__
-=:	__isub__
**:	__pow__
>:	__gt__
<:	__lt__
>=:	__ge__
==:	__eq__
!= :	__ne__

If we want to overload particular operator: we need to define these methods in the class.

*Write a program to create a class book with three attributes - title, publisher & the price. Compare the 2 given book objects based on size.

```
class Book:
    def __init__(sely):
        sely.title=" "
        sely.publisher=" "
        sely.price = 0
    def setvalues(sely,t,p, price):
        sely.title=t
        sely.publisher=p
        sely.price = price
    def display(sely):
        print ("Title:", sely.title)
        print("Publisher:", sely.publisher)
        print (" Price:", sely.price)
    def __gt__(sely, b):
        if sely.price > b.price:
            return True
        else:
            return False

b1 = Book( )
b1.setvalues("c", "Pearson", 350)
b2 = Book( )
b2.setvalue("python", "Cengage", 500)
if b1>b2:
    print ("Book", b1.price, "is more")
else:
    print( "Book" b2.price is more")
```

*WAP to overload `-=' operator to subtract 2 distance objects.

```
class Distance:
```

```
def •__init__(sey):
        ~~sey.distance~~=
        sey.km=0
    def setvalue(sey, k):
        sey.km = k
    def display(sey)
        print("Distance:", self.km)
    def __isub__(sey,d):
        sey.km = sey.km-d.km
        return sey.
d1 = Distance( )
d1.setvalues(2 30)
d2 = Distance( ).
d2.setvalues(50)
d1 -= d2
d1.display
```

26|08|2022

* isinstance & issubclass

- issubclass → Checks if derived class is a subclass of super class
- isinstance → checks if the object created is the instance of given class

*Ex

```
class student:
    ⋮
s1 = student( )
print(isinstance(s1, student))
```

* Datetime: Module: ⇒ 6 Classes.

import Datetime as dt

*date: class.

```
cd = dt.date.today( )
```

today( ) → method

```
print(cd) // → YYYY-MM-DD
cd.year
cd.Month.
cd.day.
```

* time:

```
ct = dt.time.now( )
print (ct) // → (hr:min:sec:microsec)
```

* The difference b/w 2 time objects is of timedelta object

* Datetime object:

```
cdt = dt.datetime.now ( ) ⇒ (YYYY-MM-DD) ~~(DD-MM-YYYY)~~ (hr:min:sec)

cdt = dt.datetime (2022, 08, 26).
                2022|08|26.
```

* strftime( ) ; strptime( )

%m

% A → week day name.

% b → Dec

% B → December

% d → 26 (day)

%Y → year(2018)
%y → year(18)
%H → Hour(24)
%I → Hour(12)
%P → AM/PM

Ex. cd=dt.datetime.now()
Nd =cd.strtime("%A, %B. %d, %Y")
print(cd) // 2022-08-26
print (Nd) // (Fri, Aug26,2022

* strftime() → datetime to string
* strptime() → string to datetime

* cd=dt.datetime.now()
ed = dt.datetime (2022,09,22)
diff = ed-cd
print(diff) ⇒ 14days 15hrs.. mins secs.

* TURTLE: 2 classes.
- Screen: sheet
- turtle: pen

Raw turtle
↓
Turtle (sub class)

* A turtle screen will automatically be created if its not created originally
• The screen is divided into 4 parts as of xy plane with each pixel having specific co-ordinate

* Methods.

t.forward (10)
t.backward()
t. pensize()
t. color("blue")
t. circle(50, steps=6) → controls the shape of polygon.
t.clear()

→ penup ⇒ if this is done; the turtle is off the screen and nothing is drawn.

→ pendown ⇒ opposite

```
begin_fill()
t.circle(30)
end_fill()
begin_fill()
t.circle(40); end_fill()
t.color("yellow", "black")
t.setpos(0,0)
t.clear()
for i in range(4):
        t.forward(100)
        t.left(90)
print(t.pos()) → position of the turtle
```

27/08/2022

~~t= Turtle()~~
~~t.color("yellow", "black")~~

Ex:

```
from turtle import*
import (random)
 t=Turtle()
 t.color("yellow", "black").
  t.begin_fill()
  for i in range(4):
        t.forward(200)
        t.left(90).
t.end_fill()
```

~~colors=["blue", "green",~~
~~"black", "yellow",~~
~~"orange", "red", "pink"]~~
~~for i in range(20):~~
~~400~~

*

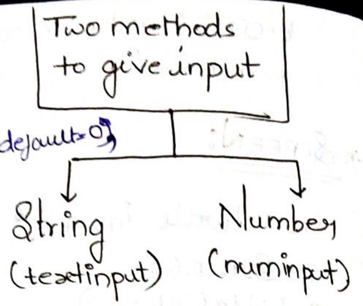
```
colors=["blue", "green", "black"]
for i in range(20):
                                   outer border
      t.color(colors[random.randint(0, len(colors)-1)],
                 colors[random.randit (0, len(colors)-1)]
                                              inner fill
      t.penup()
     t.setpos(random.randint(-200,200), ~~rando~~
                 random.randint(-200,200))
     t.pendown()
     t.begin fill()
     t.circle(random.randint(0,30))
     t.end_fill().
```

*SCREEN:

```
from turtle import* *
s=Screen()
s.title("String")
s.bgcolor("Black")
s.bgpic()  ⟶ if no pic 'nopic'
t=Turtle()
s.resetscreen()
s.reset()
s.clear()
t=Turtle()
t.fd(100
s.reset() ⟶    resets the turtle to
               initial position.
s.mode()   → 'standard' : we generaly
                 (arrow facing use standard
s.mode("logo")        right) mode.
t=Turtle()  ⟶(arrow facing upwards.
s.mode() ⟶ 'logo'
s.getshapes() ⟶ ['arrow', 'blank',
                 'circle', 'classic', 'square',
                 'triangle', 'turtle']

s.bye()
```

```
s = Screen()
s.textinput("cse", "name")
            title   prompt
s.numinput("cse", "age", default=0,
minval=10, maxval=20)
```



* PANDAS:

- Analyse, Clean, manipulate the data
- Data is generally stored in csv (comma separated values)

* Represent data:

- Series ( )
- Dataframe ( ) → 2D

} → parameters are list, dictionaries, csv, json.

```
* import pandas as pd.
  a = [1, 7, 2, 4, 8]
  myvar = pd.series(a, index
                 ['a', 'b', 'c', 'd', 'e'])
  print(myvar)
```

default index

```
0  1
1  7
2  2
3  4
4  8
```

```
a  1
b  7
c  2
d  4
e  8
dtype = int64
```

```
data = { 'calories': [420, 300, 390],
         'duration': [50, 40, 30]}
print(data)
```

→ for rows & columns.

```
df = pd.read_csv('data.csv')
print(df.to_string)
print(df)

pd.options.display.max_rows = 'value'
```

print(df.head(10)) → parameter specify the no. of rows to extract from begin
↳ default: prints only first 5 rows.

(df.tail(10)) → default extracts last 5 rows.
→ parameter no. of rows to extract from end.

→ df.dropna() → removes the empty cell and returns new data.

→ df.fillna(130, inplace=True) → modifying/correcting the data

→ df["calories"].fillna(130, inplace=True) → Column.

```
df["calories"].mean()
              .median()
              .~~mean~~
df["calories"].mode()[0]
```

30/08/2022

```
import pandas as pd.
df = pd.read_csv('data.csv')
df['Date'] = pd.to_datatime(df['Date'])
Print (df.to_string()).
```

if ~~df.loc[x,'Duration']>120~~
drop~~(x~~

import matplotlib.pyplot
plt.pie(y)

02|09|2022

* dic1 = {x:2*x for x in range(1,5)}
  print(dic1)

* When we want key-value pairs and use keys to retrieve data

* Operator overloading.

* When object is created: the constructor is executed.

* Only explanation/pargraph on Turtle, Pandas.

* Sample code in matplotlib – Significance

* Exception handling – Own exception creating/ raise, re-raise the exceptions.

* → Assertion

  – ~~Atleast~~ try block should atleast hav either one except/finally block.

* Fill,

*→ WAP that has a set of numbers 1 to 5 in figures and words in a dictionary. Define another dictionary that has list of 1 to 5 words and the corresponding roman numerals. Display the numbers 1 to 5 in figures, words and their corresponding roman numbers.

~~d={}~~
for i ~~in range(1,6):~~
    ~~d[i] =input("En~~

* x = np.array([1,2,3,4])
  y = x*2
  plt.plot(x,y)
  plt.xlabel("x-axis")
  plt.ylabel("y-axis")
  plt.title("Graph")
  plt.show()

x~~x**x~~ to

* Explain how exception is raised and re-raised using an example.
  ↳ calling raise in except block