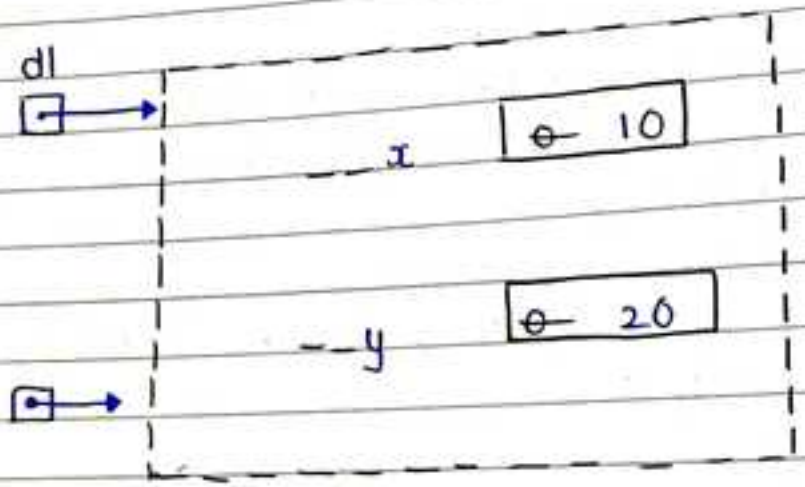
P353.py

```
class Base: # parent
    def __init__(me):
        me.__x = 0
    # end: init
    def input(me):
        print('enter x: ', end = '')
        me.__x = int(input())
    # end: input.
    def output(me):
        print(f'x: {me.__x}')
# end: output
# end: Base.
class Derived(Base): # child
    def __init__(me):
        Base.__init__(me)
        me.__y = 0
    # end: init
    def input(me): # overided fn: (base + derived)
        Base.input(me)
        print('enter y: ', end = '')
        me.__y = int(input())
    # end: input
    def output(me): # overided fn.
        Base.output(me)
        print(f'y: {me.__y}')
    # end: output
# end: Derived
def main():
    d1 = Derived()
    d1.input()
    d1.output()
```

DATE

end: main

main()



%p

enter x : 10 ↵
enter y: 20 ↵
x : 10
y: 20

DATE

P359.py

```
class Base: # parent
    def funBase(me):
        print('In funBase')
    # end: funBase
# end: Base
```

ccc

derived class obj. can call fns from base class and derived class

---

base class obj. can call fns from base class
base class obj. cannot call fns from derived class

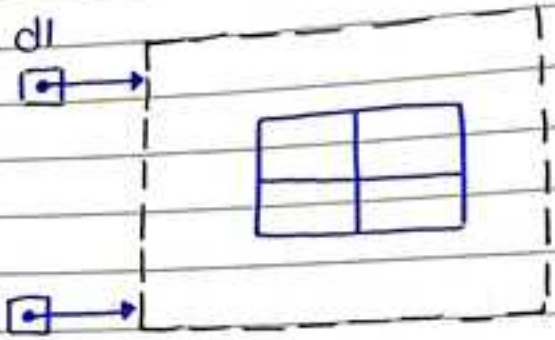
ccc

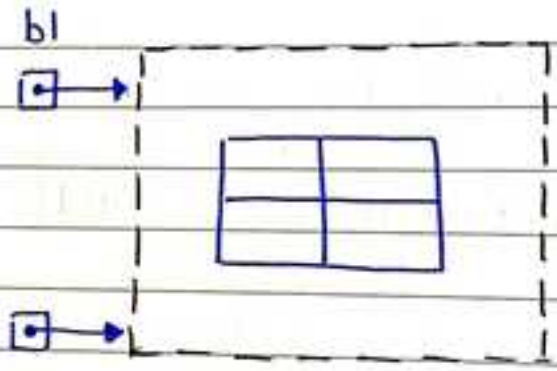
```
class Derived(Base): # child
    def funDerived(me): # non-overided
        print('in funDerived')
    # end: funDeried
# end: Derived
def main():
    d1 = Derived()
    d1 = funBase()
    d1 = funDerived()
    b1 = Base()
    b1 = funBase()
```


PAGE

bl.funDerived() # error
end: main

main()





%p

In funBase
In funDerived
In funBase

P360.py

```
class Base:
    def __init__(me):
        me.__a = 10
        me._b = 20
        me.c = 30
    # end: init
    def output(me):
        print(me.__a)
        print(me._b)
        print(me.c)
    # end: output
# end: Base
```

*	Inside the class of defⁿ	Outside the class defⁿ: Derived class / child class (Inheritance)	Outside the class defⁿ: Non-Derived class / Non-child class / Non-member fn (main)
private (__var)	✓	✗	✗
protected (_var)	✓	✓	✗
public (var)	✓	✓	✓

DATE

Access Specifiers

private

Rules

(__ variable)

↓

protected

Advice

(_ variable)

public

Rules

(variable)

DATE

P361.py

```
class Base:
    def __init__(me):
        me.__a = 10
        me._b = 20
        me.c = 30
    #end: init
# end: Base
class Derived (Base):
    def output(me):
        # print (me.__a) # error
        print (me._b)
        print (me.c)
    # end: output
# end: Derived
```

P362.py

```
class Base:
    def __init__(me):
        me.__a = 10
        me._b = 20
        me.c = 30
    # end: init
# end: Base
class X: # non-dervied
    def output(me):
        ob = Base()
        # print (ob.__a) # error
        # print (ob._b) # not advised
        print (ob.c)
    # end: output
```


PAGE

DATE

#end: x

P363.py

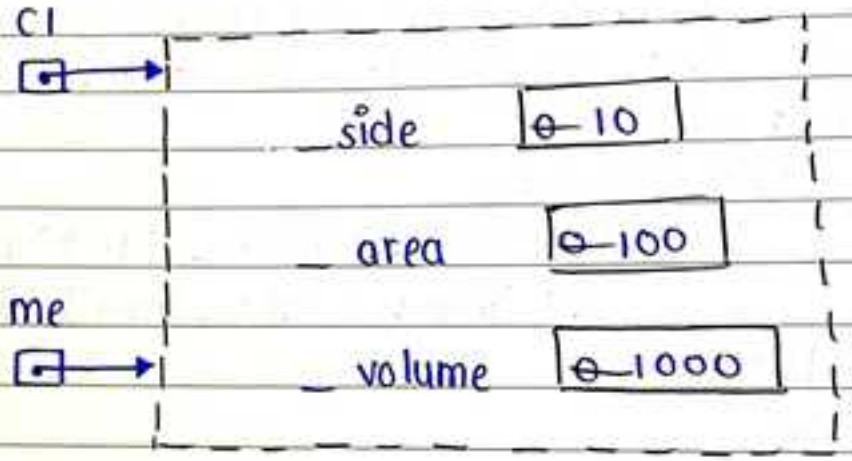
```
class Base:
    def __init__(me):
        me.__a = 10
        me._b = 20
        me.c = 30
    #end: init
#end: Base
def main():
    ob = Base()

    """
    print (ob.__a) # error
    print (ob._b) # Not advisable
    """
    print (ob.c)

#end: main

main()
```

DATE

P364.py

```
class Square:
    def __init__(me):
        me._side = 0
        me._area = 0
    # end: init
    def input(me):
        print('enter side :', end = '')
        me._side = int(input())
        me._area = me._side ** 2
    # end: input
    def output(me):
        print(f'side : {me._side}\n'
              f'Area : {me._area}')
    # end: output
# end: square.
class cube(square):
    def __init__(me):
        square.__init__(me)
        me._volume = 0
    #end: init
    def input(me):
        square.input(me)
        me._volume = me._area * me._side
    #end: input
    def output(me):
        $quare.output(me)
        print(f'Volume : {me._volume}')
    # end: output
# end: cube
```

```
def main():
    c1 = cube()
    c1.input()
    c1.output()
#end: main

main()
```



o/p

enter side: 10 ↵
side: 10
Area: 100
Volume: 1000

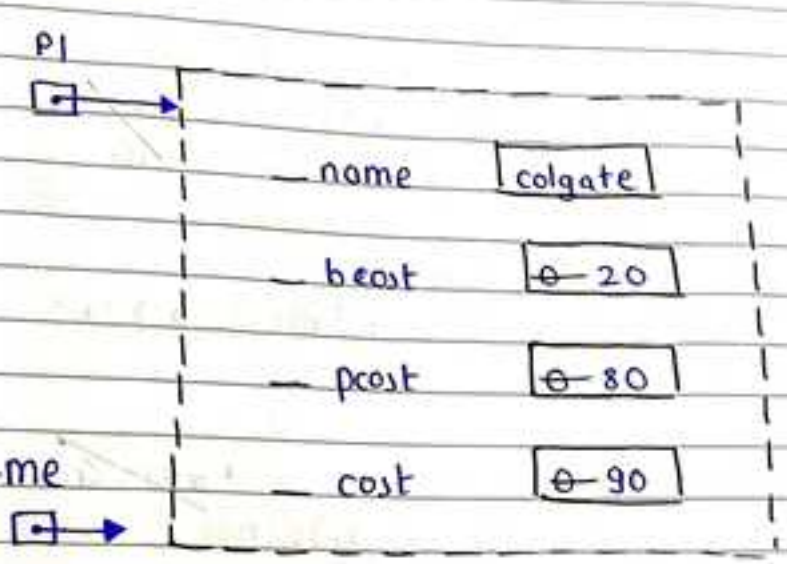
DATE

P365.py

```
# Multiple heritance
class Company:
    def __init__(me):
        me._name = ''
    #end: init
    def input(me):
        print('enter company :')
        me._name = input()
    # end: input
    def output(me):
        print(f'Company : {me._name}')
    # end: output
#end: Company.
class Brush:
    def __init__(me):
        me._bcost = 0
    # end: init
    def input(me):
        print('enter brush cost :')
        me._bcost = int(input())
    # end : input
    def output(me):
        print(f'Brush cost: {me.bcost}')
    #end: output
# end: Brush.
class Paste:
    def __init__(me):
        me._pcost = 0
    #end : init
    def input(me):
        print('enter paste cost :')
        me._pcost = int(input())
```

PAGE

```
    #end: input
    def output (me):
        print (f' Paste cost : {me._pcost}')
    #end: output
#end: Paste
class Pack (Company, Brush, Paste):
    def __init__(me):
        Company.__init__(me)
        Brush.__init__(me)
        Paste.__init__(me)
        me._cost = 0
    #end: init
    def input (me):
        Company.input (me)
        Brush.input (me)
        Paste.input (me)
        me._cost = 0.9 * (me._bcost + me._pcost)
    #end: input
    def output (me):
        Company.output (me)
        Brush.output (me)
        Paste.output (me)
        print (f' Pack cost: {me._cost}' )
    #end: output
#end: Pack
def main ():
    p1 = Pack ()
    p1.input()
    p1.Output()
#end: main

main()
```



o/p

enter company :
colgate ↵
enter brush cost :
20 ↵
enter paste cost :
80 ↵
Company Colgate
Brush cost : 20
Paste cost : 80
Pack cost : 90

P366.py

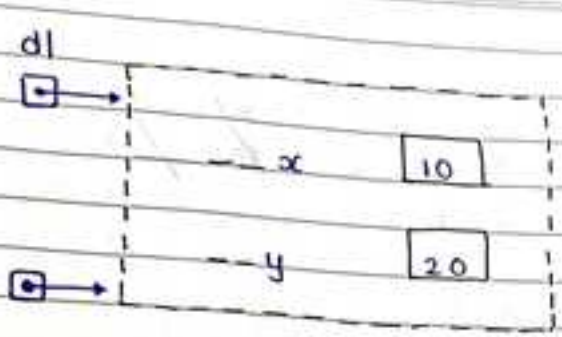
```
class Base:
    def __init__(me, x):
        me.__x = x
    # end: init
    def __str__(me):
        return f'x: {me.__x}\n'
    # end: str
# end: Base
class Derived(Base):
    def __init__(me, x, y):
        Base.__init__(me, x)
        me.__y = y
    # end: init
    def __str__(me):
        return (Base.__str__(me) +
                f'y: {me.__y}')
    # end: str
# end: Derived
def main():
    d1 = Derived(10, 20)
    print(d1)
    # print(d1.__str__())
# end: main

main()
```

x/10

'x:10\n'

x/10 y/20

'x:10\ny:20'

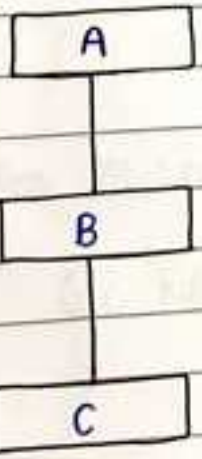



```
%p
x : 10
y : 20
```

Multi - Level Inheritance

P367.py

```
class A:                                    x/10
    def __init__(me ,x):
        me.__x = x
    #end : init
    def output(me):
        print (f'x : {me.__x}')
#end: A
```

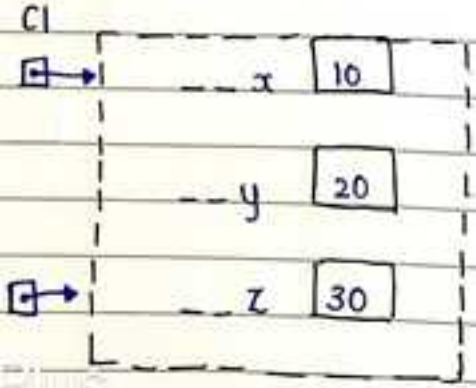


```
class B(A):
    def __init__(me, x, y):
        A.__init__(me, x)
        me.__y = y
    # end: init
    def output(me):
        A.output(me)
        print(f'y {me.__y}')
    # end: output
# end: B
class C(B):
    def __init__(me, x, y, z):
        B.__init__(me, x, y)
        me.__z = z
    # end: init
    def output(me):
        Be.output(me)
        print(f'z {me.__z}')
    # end: output
# end: C
def main():
    C1 = C(10, 20, 30)
    C1.output()
#end: main
```

x/10 y/20

x/10 y/20 z/30

main()



DATE

o/p

x : 10

y : 20

z : 30

P368.py

```
class Student:
    def school(me): pass
# end: Student
def main():
    S1 = student()
    S1.school()
# end: main

main()
```

* ABC

↓

Abstract

Base

class

Abstract incomplete

(IA)

Non Abstract complete

(C NA)

PAGE

P369.py

```
from abc import ABC, abstractmethod

# abc : module
# ABC : class
# abstractmethod : function

class Student (ABC) :  # I A
    @ abstractmethod
    def school (me): pass
#end: Student
def main():
    S1 = Student () #error
    :
#end: main

main()
```

⊙ class that contains (IC) abstract method is abstract class (IC)

⊙ object of abstract class cannot be defined

DATE

P370.py

```
from abc import ABC, abstractmethod

class student (ABC): # IC: A
    @abstractmethod
    def school(me): pass  # IC: A
# end: student
class Amit (student): # C: NA
    def school(me): # C: NA
        print ('BV')
    # end: school
# end: Amit
class Raj (student): # IC: A
    def school(me):
        print ('IMS')
    # end: school
    # school : IC: A
# end: Raj
def main():
    # ob1 = student () # error
    ob2 = Amit()
    ob2.school()
    # ob3 = Raj() # error
# end: main

main()
```

o/p

BV

PAGE

P371.py

```
from abc import ABC, abstractmethod

class Country (ABC):
    @abstractmethod
    def capital (me): pass   # IC : A
    @abstractmethod
    def currency (me): pass   # IC : A
#end: country
class India (country):   # C: NA
    def capital (me): print ('India : New Delhi')
    def currency (me): print ('India : Rupee')
    # capital : C : NA
    # currency : C : NA
#end: India
class Japan (country) : # IC : A
    def capital (me): print('Japan : Tokyo')
    #capital : C : NA
    # currency : IC : A
# end: Japan
def main():
    # ob1 = Country () #error
    ob2 = India()
    ob2 . capital()
    ob2 . currency()
    # ob3 = Japan() #error
#end: main

main()
```

o/p

```
India: New Delhi
India: Rupee
```

DATE

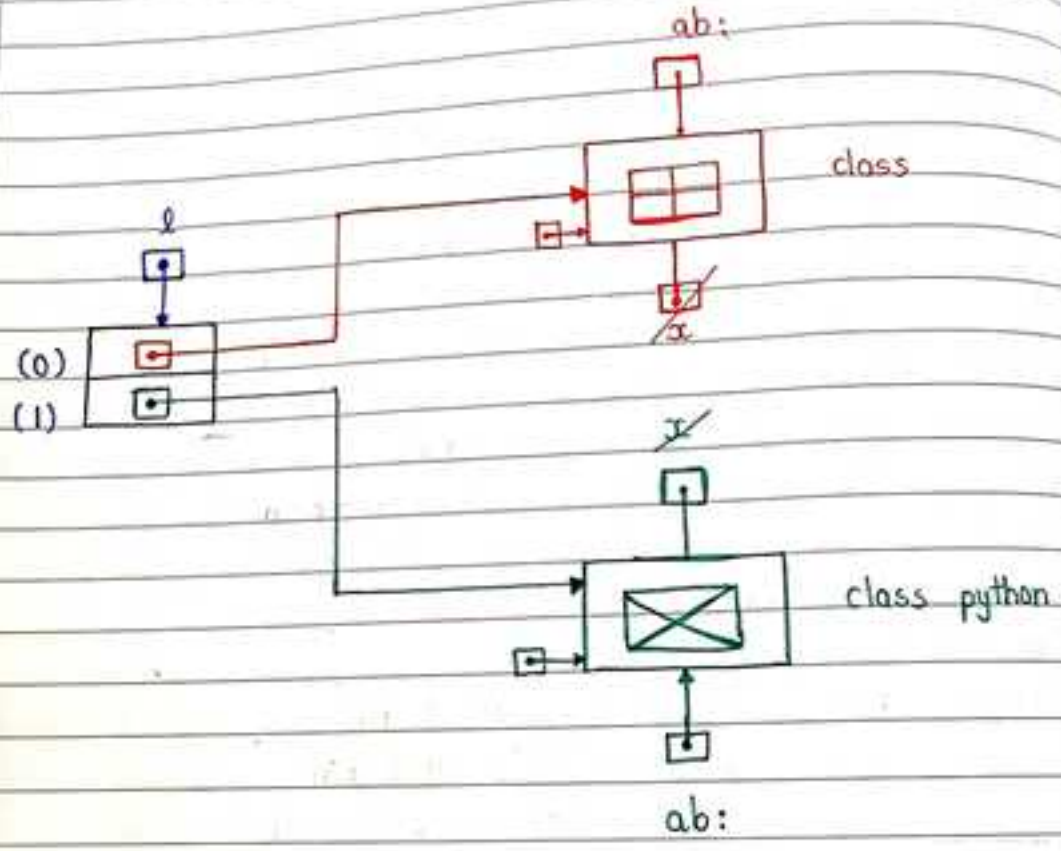
Run - Time Polymorphism

P373.py

```
from abc import ABC : abstractmethod

Class Course (ABC): # IC : A
    @abstractmethod
    def duration (): pass  # IC:A
# end: Course
class C (course) : # C : NA
    def duration() : # C:NA
        print ('C : 3 months')
    # end: duration
# end: C.
Class Python (course): # C: NA
    def duration () : # C:NA
        print ('Python: 5 months')
    # end: duration
# end: python
def main():
    ob1 = C()
    ob2 = Python()
    l = [ob1, ob2]
    for x in l: #
        x.duration()
    #end:for
# end: main

main()
```


PAGE

DATE



C : 3 months

Python : 5 months

DATE

P374.py

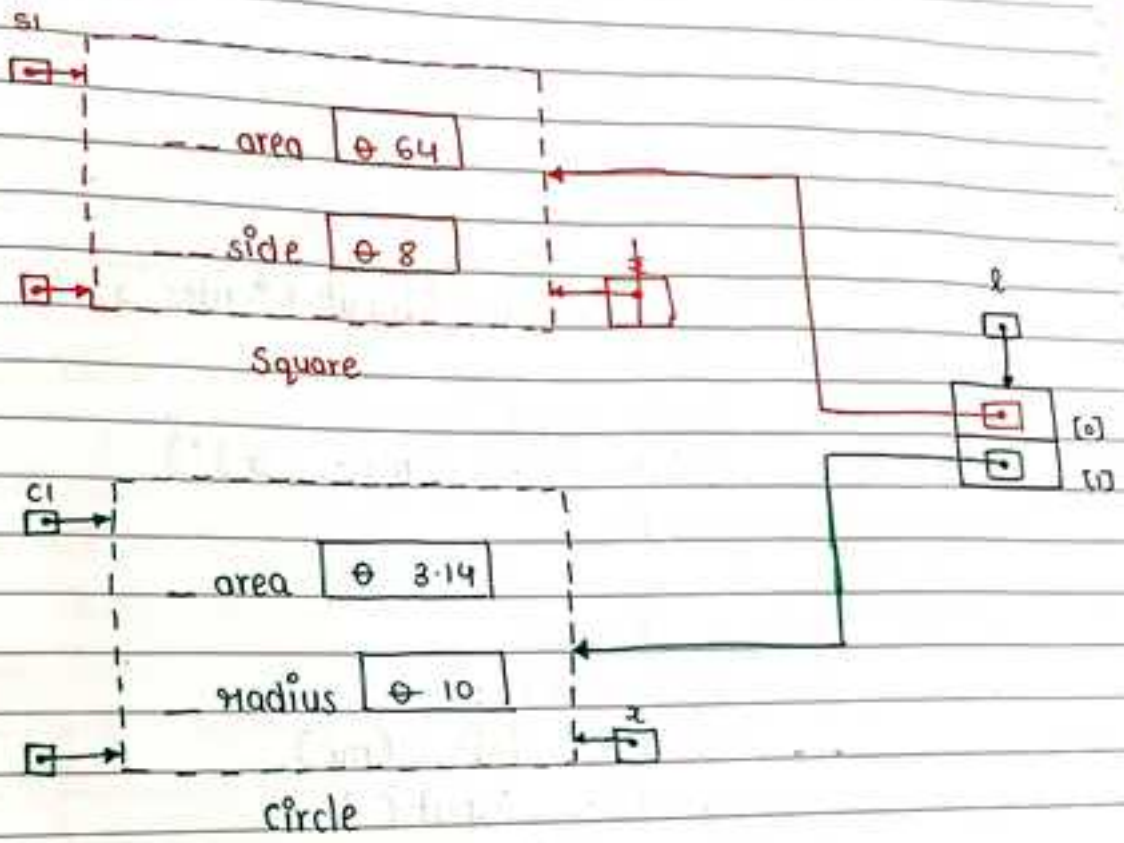
```
from abc import ABC, abstractmethod
class GeoFig (ABC) : # IC : A
    def __init__(me):
        me._area = 0
    # end: init
    @abstractmethod
    def input(me): pass # IC:A
    @abstractmethod
    def output (me): pass # IC : A
# end: GeoFig.
class Square (GeoFig): # C: NA
    def __init__(me):
            GeoFig.__init__(me)
            me._side = 0
    # end: init
    def input(me): # C:NA
            print ('enter side : ', end = '' )
            me._side = int( input())
            me._area = me._side ** 2
    #end: input.
    def output (me): # C : NA
        print (f'side : {me._side}'
        f'\n Area : {me._area}')
    #end: output
# end: Square
class Circle (GeoFig): # C:NA
    def __init__(me):
        GeoFig.__init__(me)
        me.radius = 0
    #end: init
```

PAGE

classmate

4:41 pm

```
        def input(me): # C: NA
            print('enter radius :', end = '')
            me._radius = int(input())
            me._area = 3.14 * me._radius * *2
        #end: input
        def output(me): # C: NA
            print(f' Radius : {me._radius}'
            f'\n Area : {me._area}')
        #end : output
#end : Circle.
def main():
    S1 = Square()
    C1 = Circle()
    l = [S1, C1]
    # l = [Square(), Circle()]
    ' ' '
    S1.input()
    S1.output()
    C1.input()
    C1.output()
    ' ' '
    for x in l: #
        x.input()
        x.output()
    #end: for
#end: main

main()
```



enter side : 8 ↵
side : 8
Area : 64

enter radius : 10 ↵
Radius : 10
Area : 3.14

P375.py

```
class Base:
    def __init__(me):
        me.__x = 0
    # end: init
    def input(me):
        me.__x = int(input('enter x:'))
    # end: input
    def output(me):
        print(f'x: {me.__x}')
    # end: output
# end: Base
class Derived(Base):
    def __init__(me):
        # Base.__init__(me)
        super().__input()
        me.__y = 0
    # end = init
    def input(me):
        # Base.input(me)
        super().input()
        me.__y = int(input('enter y:'))
    # end: input
    def output(me):
        # Base.output(me)
        super().output()
        print(f'y: {me.__y}')
    # end: output
#end: Derived
```

DATE

```
# super : superior.

def main():
    d1 = Derived()
    d1.input()
    d1.output()
# end: main

main()
```

DATE

P376.py

```
def capital():
    print('New Delhi')
# end: capital
def india():
    print('India: ', end='')
    capital()
# end: india

def main():
    india()
    capital()
# end: main
main()
```

o/p

```
India: New Delhi
New Delhi
```

DATE

p377.py

```
def india() : # outer fn
    def capital : # inner fn
        print ('New Delhi')
    # end: capital
    print ('India : ', end = '')
    capital()
# end: India

def main():
    india()
    # capital() # error
# end: main

main()
```

o/p

India : New Delhi

DATE

P378.py

```
# inner fn can use vars of outer fn

def calculate (x):
    def square():
        print(f'square.{x**2}')
    # end: square
    def cube():
        print(f'cube:{x**3}')
    # end: cube
    square()
    cube()
# end: calculate

def main():
    calculate(5)
# end: main

main()
```

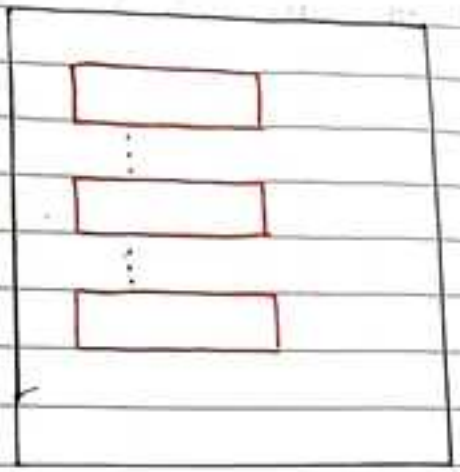
calculate

x
5

o/p

```
Square: 25
Cube: 125
```

def ofn():



def ofn():

ofn : outer function

ifn : inner function

```
def ifn()
    [ ]
#end: ifn
:
ifn()
:
ifn()
:
ifn()
:
```

P319.py

```
def border (fn):
    def hash (n):
        print ('#' * len (n))
        fn(n)
        print ('#' * len (n))
    # end: hash
    return hash
# end: border

@ border
def greet (name):  # border (greet)
    print (name)
# end: greet

def main():
    greet ('sam')     # hash ('sam')
    greet ('rahul')   # hash ('rahul')
# end: main

main()
```

greet

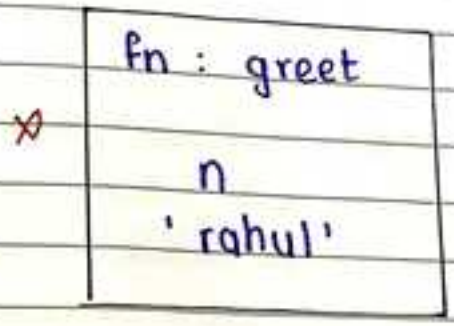
Name
'sam'

Name
'rahul'

border

fn: greet
n
'sam'

DATE



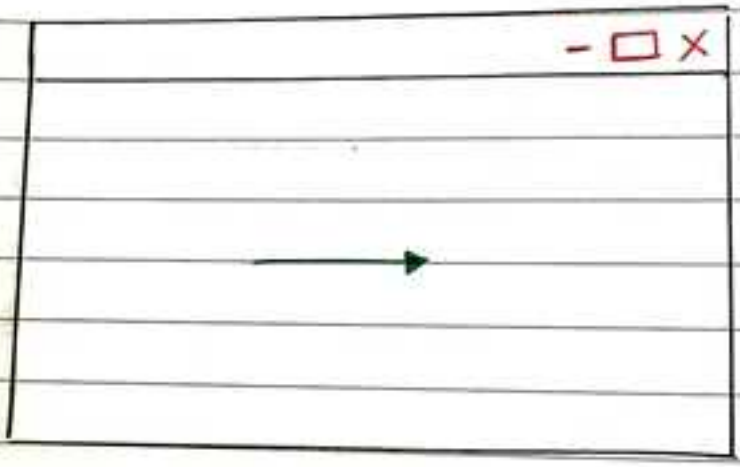
# # #
s a m
# # #

# # # # #
r a h u l
# # # # #

DATE

P380.py

```
from turtle import *
t = Turtle()
t.forward(100) # pixel  picture element
mainloop()
```



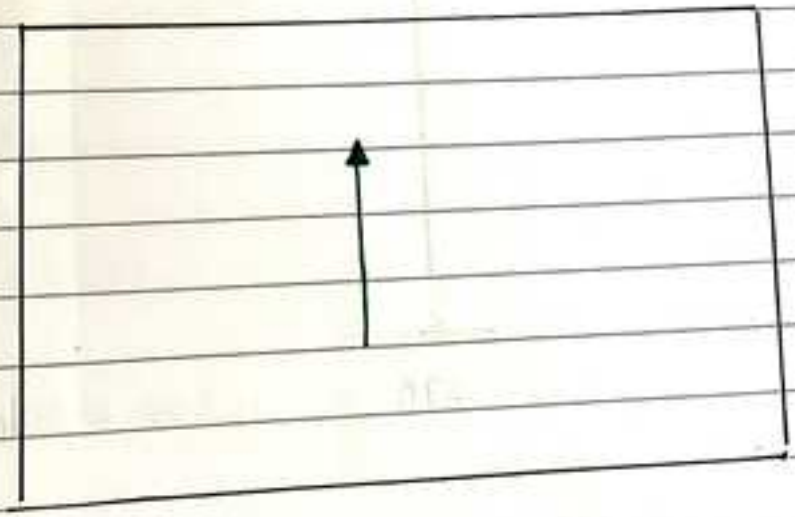
24

$\frac{360}{24} : 15°$

DATE

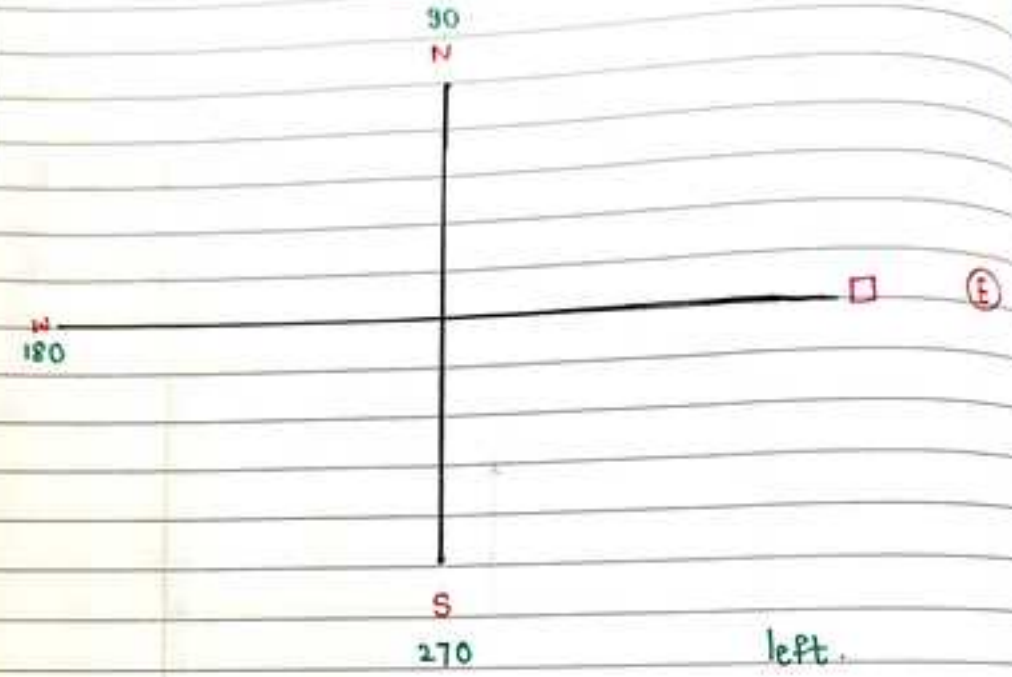
P381.py

```
from turtle import *
t = Turtle()
t.left(90)
t.forward(100)
mainloop()
```

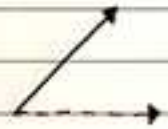


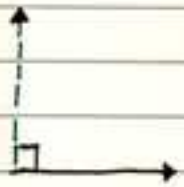
DATE

Turtle 1969



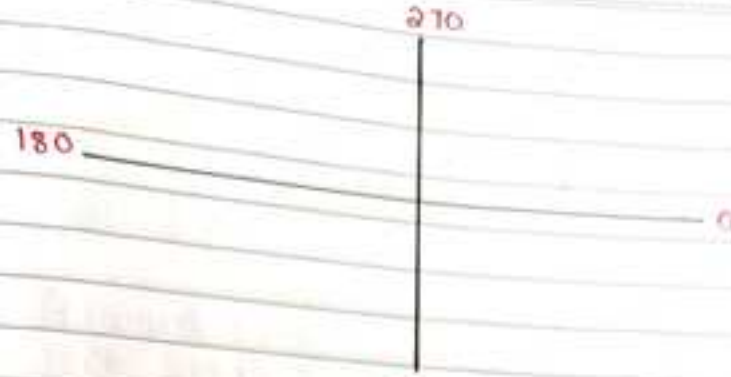
left.





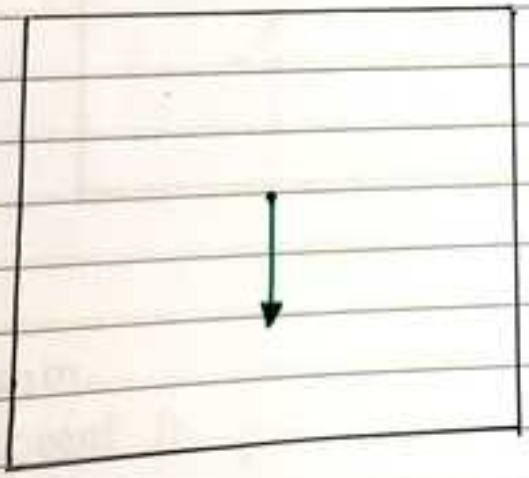
classmate

PAGE



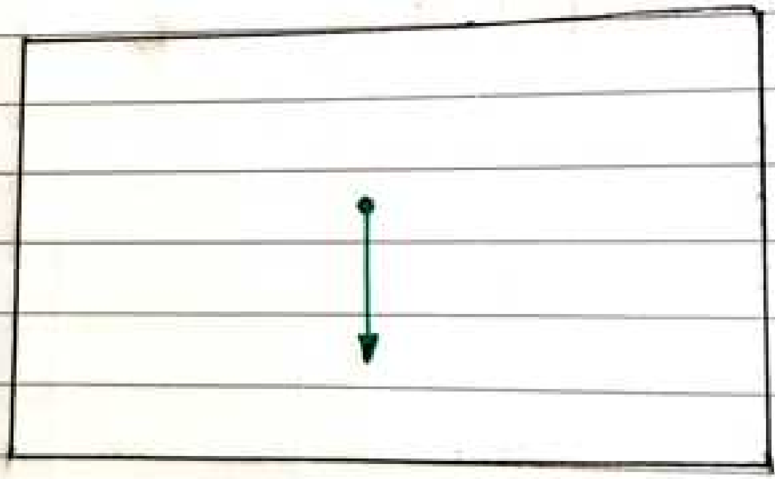
P 382.py

```
from turtle import *
t = Turtle()
t.left(270)
t.forward(100)
mainloop()
```



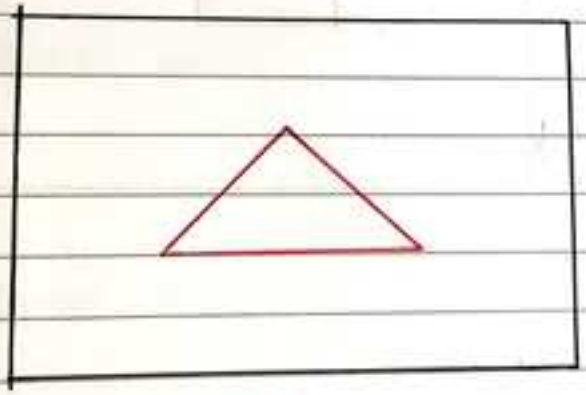
P383.PY

```
from turtle import *

t= Turtle ()

t. right (90)

t. forward (100)

mainloop()
```



P384.py

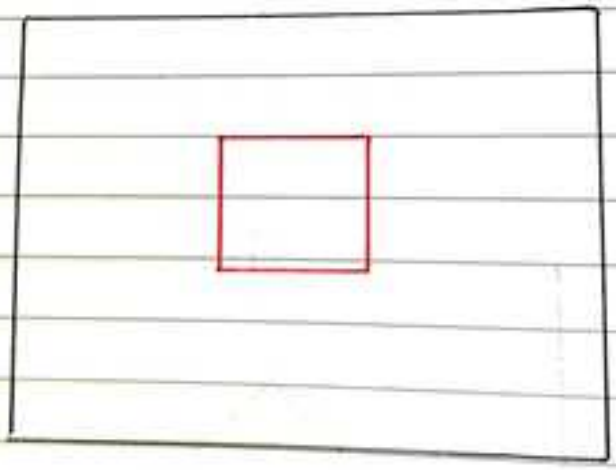
```
from turtle import *
t = Turtle()
t.forward(100)
t.left(120)
t.forward(100)
t.left(120)
t.forward(100)
t.left(120)
mainloop()
# loop(use)
```



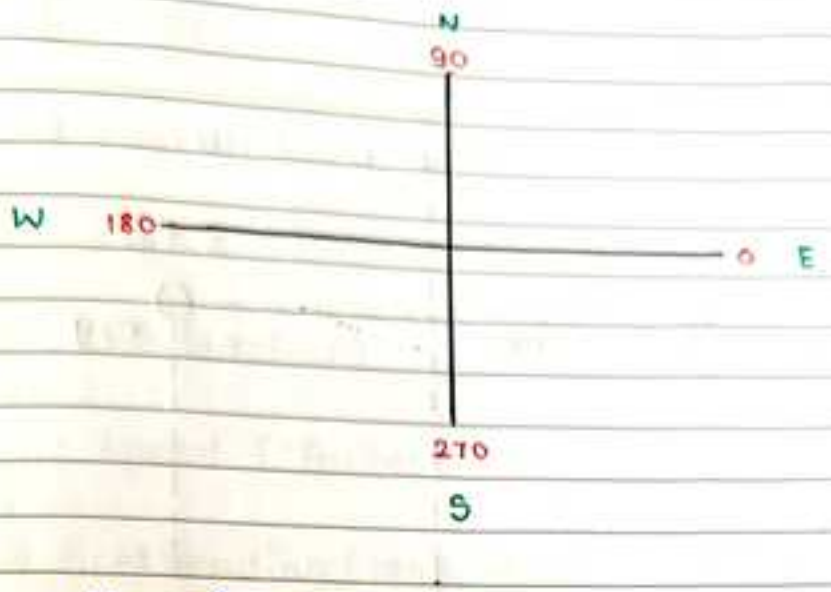
Ass: # solve this pgm
using backward fn.

P385.py

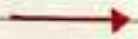
```
from turtle import *

t= Turtle()

t.pencolor ('red')
for _ in range (4): # 0 1 2 3
    t.forward (100)
    t.left (90)  # 360/4
# end : for

mainloop()
```



DATE



- setheading (0) E

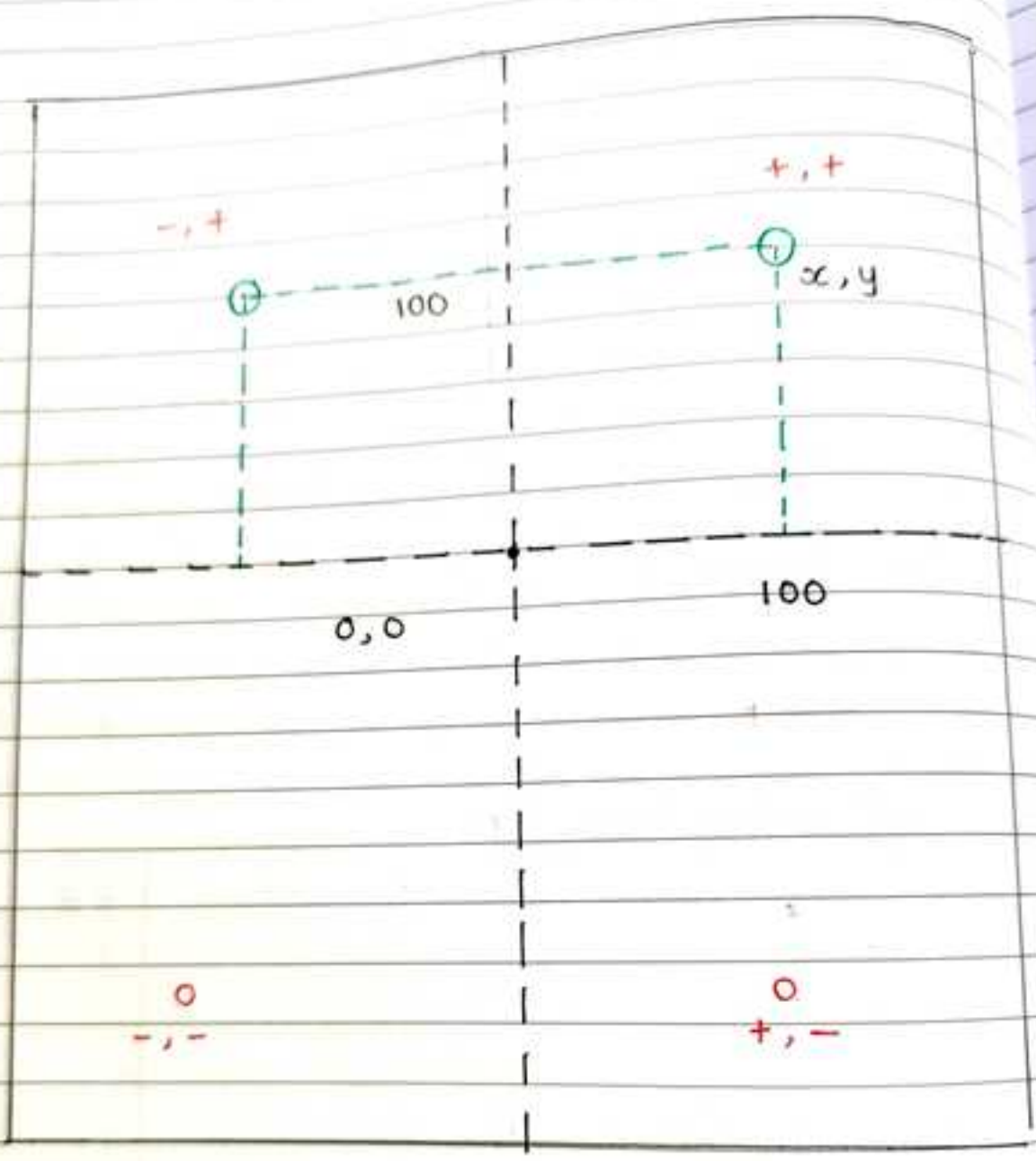


- setheading (90) N

- setheading (180) W

- setheading (270) S

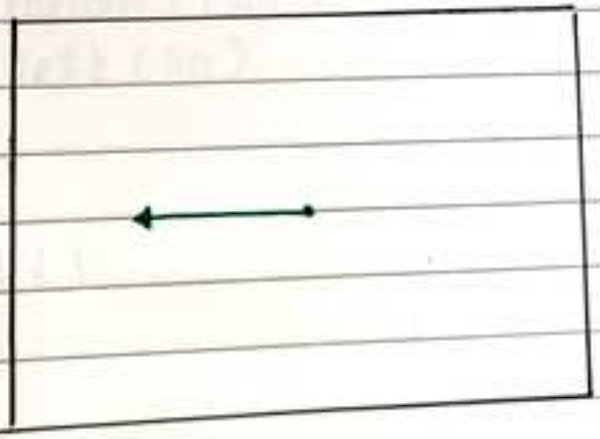
DATE



DATE

P386.py

```
from turtle import *
t = Turtle ()
t.hideturtle ()
t.speed ('fastest')
t.setheading (180) # w
t.forward (100)
mainloop()
```



P387.PY

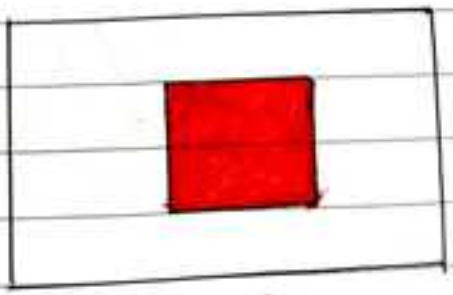
```
from turtle import *

t = Turtle()

t.up()
t.goto(100,100)
t.down()
t.circle(25) # radius

t.up()
t.goto(-100,100)
t.down()
t.circle(25)

mainloop()
```

DATE

P388.py

```
from turtle import *
t = Turtle()
t.speed('fastest')
t.pencolor('blue')
t.fillcolor('red')
t.begin_fill()
for _ in range(4):
    t.forward(100)
    t.left(90)
# end : for
t.end_fill()
mainloop()
```

fastest
fast
normal
slow
slowest



PAGE

p389.py

```
from turtle import *

t = Turtle()

for _ in range(3):
    t.forward(100)
    t.left(120)
#end: for

t.up()
t.forward(200)
t.down()

for _ in range(4):
    t.forward(100)
    t.left(90)
#end: for

mainloop()
```