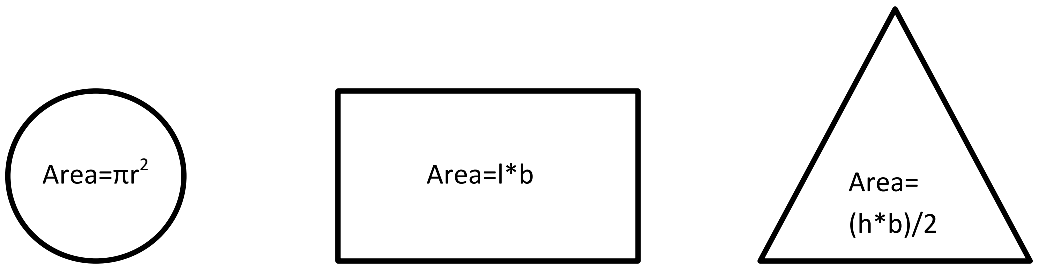
Python fundamentals day 60

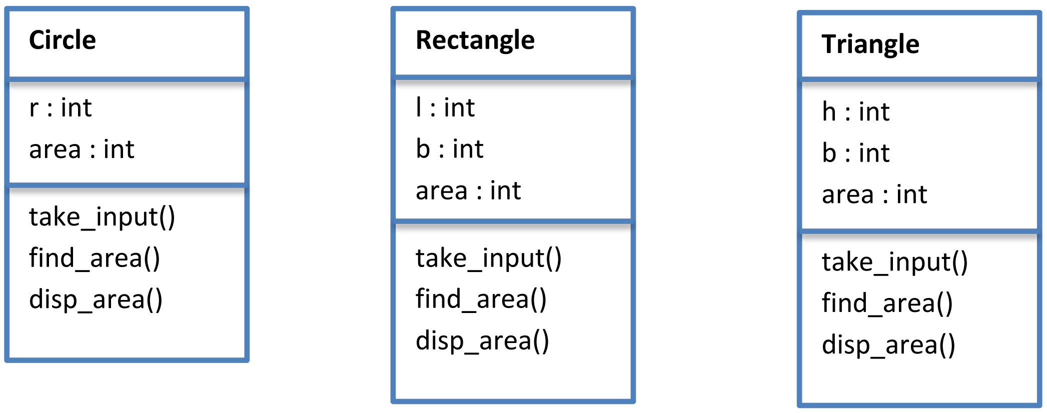
Today's agenda

- Abstraction contd
- Problems on abstraction

Abstraction contd

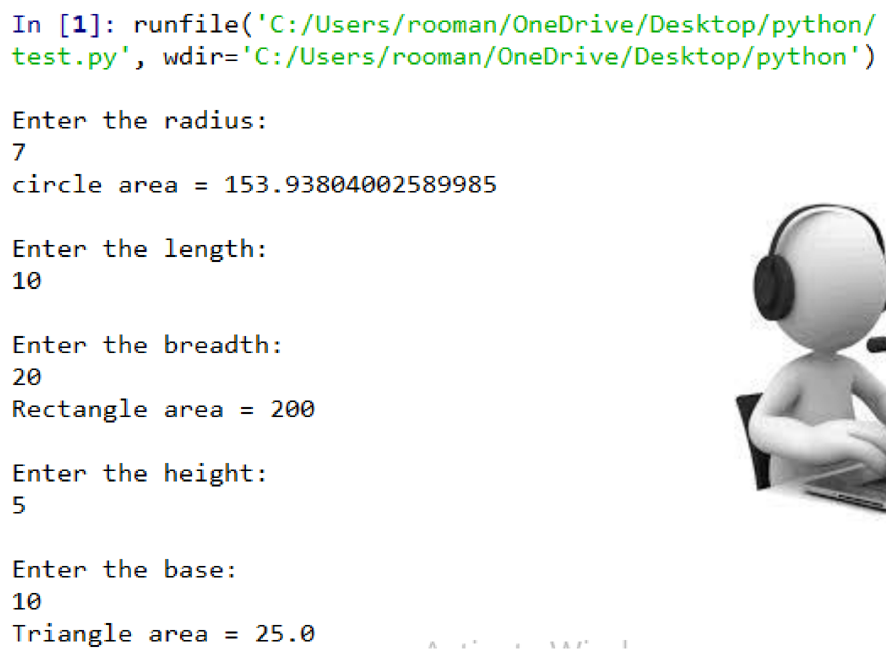
Let us take an example of shapes and try to understand when a method must be abstract, incomplete or a concrete method





```python
from math import pi
class Circle:
    def __init__(self):
        self.r=0
        self.area=0

    def take_input(self):
        self.r=int(input("Enter the radius: \n"))

    def find_area(self):
        self.area=pi*self.r**2

    def disp_area(self):
        print(f'circle area = {self.area}')

class Rectangle:
    def __init__(self):
        self.l=0
        self.b=0
        self.area=0

    def take_input(self):
        self.l=int(input("Enter the length: \n"))
        self.b=int(input("Enter the breadth: \n"))

    def find_area(self):
        self.area=self.l*self.b

    def disp_area(self):
        print(f'Rectangle area = {self.area}')

class Triangle:
    def __init__(self):
        self.h=0
        self.b=0
        self.area=0


    def take_input(self):
        self.h=int(input("Enter the height: \n"))
        self.b=int(input("Enter the base: \n"))

    def find_area(self):
        self.area=(self.h*self.b)/2

    def disp_area(self):
        print(f'Triangle area = {self.area}')

def main():
    c=Circle()
    r=Rectangle()
    t=Triangle()

    c.take_input()
    c.find_area()
    c.disp_area()

    r.take_input()
    r.find_area()
    r.disp_area()

    t.take_input()
    t.find_area()
    t.disp_area()

main()
```

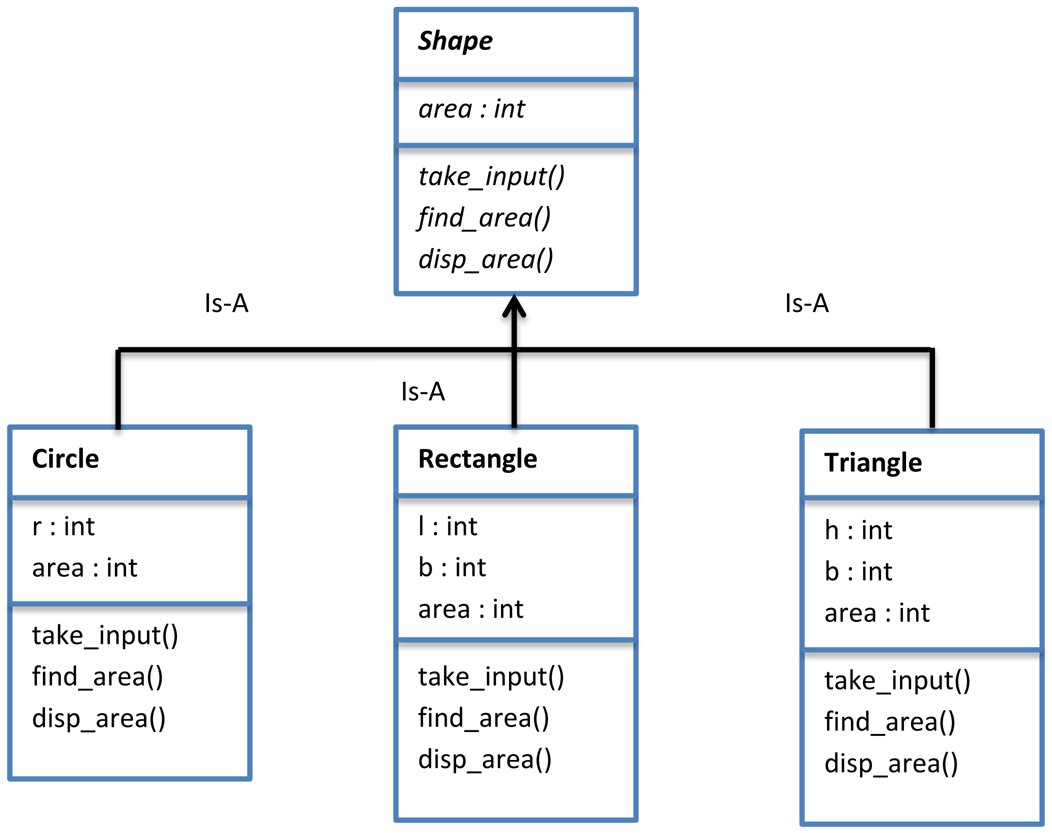
Output:



```
In [1]: runfile('C:/Users/rooman/OneDrive/Desktop/python/
test.py', wdir='C:/Users/rooman/OneDrive/Desktop/python')

Enter the radius:
7
circle area = 153.93804002589985

Enter the length:
10

Enter the breadth:
20
Rectangle area = 200

Enter the height:
5

Enter the base:
10
Triangle area = 25.0
```

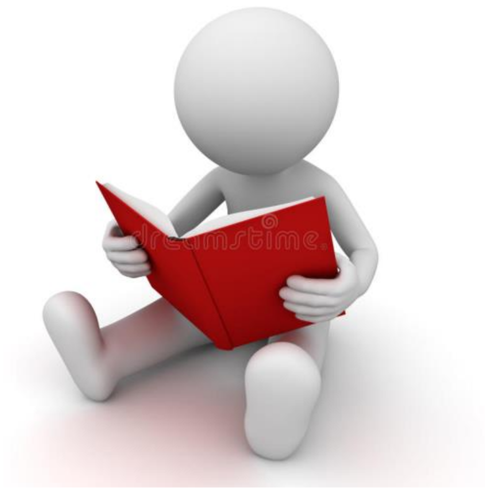
But we see the above code is not object oriented, and does not exhibit any features of object orientation like inheritance, polymorphism and abstraction.

Let us try to code using these features



```python
from math import pi
from abc import ABC,abstractmethod

class Shape(ABC):

    def __init__(self):
        self.area=0

    @abstractmethod
    def take_input(self):
        pass

    @abstractmethod
    def find_area(self):
        pass

    @abstractmethod
    def disp_area(self):
        pass

class Circle(Shape):
    def __init__(self):
        self.r=0


    def take_input(self):
        self.r=int(input("Enter the radius: \n"))

    def find_area(self):
        self.area=pi*self.r**2

    def disp_area(self):
        print(f'circle area = {self.area}')

class Rectangle(Shape):
    def __init__(self):
        self.l=0
        self.b=0
        super().__init__()

    def take_input(self):
        self.l=int(input("Enter the length: \n"))
        self.b=int(input("Enter the breadth: \n"))

    def find_area(self):
        self.area=self.l*self.b

    def disp_area(self):
        print(f'Rectangle area = {self.area}')
```

```python
class Triangle(Shape):
    def __init__(self):
        self.h=0
        self.b=0
        super().__init__()

    def take_input(self):
        self.h=int(input("Enter the height: \n"))
        self.b=int(input("Enter the base: \n"))

    def find_area(self):
        self.area=(self.h*self.b)/2

    def disp_area(self):
        print(f'Triangle area = {self.area}')

def geometric_shapes(ref):
    ref.take_input()
    ref.find_area()
    ref.disp_area()

def main():
    c=Circle()
    r=Rectangle()
    t=Triangle()

    geometric_shapes(c)
    geometric_shapes(r)
    geometric_shapes(t)

main()
```



Output:

```
In [3]: runfile('C:/Users/rooman/OneDrive/Desktop/python/
test.py', wdir='C:/Users/rooman/OneDrive/Desktop/python')

Enter the radius:
7
circle area = 153.93804002589985

Enter the length:
10

Enter the breadth:
20
Rectangle area = 200

Enter the height:
5

Enter the base:
10
Triangle area = 25.0
```

Problems on abstraction

Example 1: Let's try a larger application like fund transfer before moving on to our application. So, in fund transfer there are three types NEFT/IMPS/PTGS.

We can create an abstract class FundTransfer and extend it in the child classes. Create an abstract method transfer and implement in all the child classes.

Create an abstract class FundTransfer with following attributes, accountNumber:

int, balance : float and following methods,

validate(amount) : to check if the accountNumber is 10 digits, transfer the amount in non-negative and less than balance, and return true, if not false

transfer (amount) : abstract method with no definition

Create a class NEFTTransfer which extends FundTransfer and implements transfer method,

transfer(amount) : Check if transfer amount + 5% of transfer amount is less than balance, then subtract transfer amount and 5% service charge from balance and return true, if not return false

Create a class IMPSTransfer which extends FundTransfer and implements transfer method,

transfer(amount) : Check if the transfer amount + 2% of transfer amount is less than balance, then subtract transfer amount and 2% service charge from balance and return true, if not return false

Create a class RTGSTransfer which extends FundTransfer and implements Trandfer method,

transfer(amount) : Check if transfer amount is greater than Rs.10000, then subtract the transfer amount from balance and return true, if not return false

Add appropriate getters/setters, constructors with super() to create objects

Note: Print "Account number or transfer amount seems to be wrong" if validate function returns false.

Print "Transfer could not be made" if transfer function returns false.

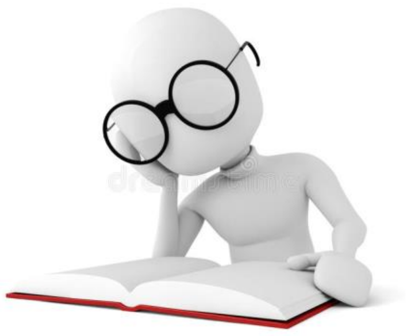
Sample Input/Output:

Enter your account number: 1234567890

Enter the balance of the amount: 10000

Enter the type of transfer to be made:

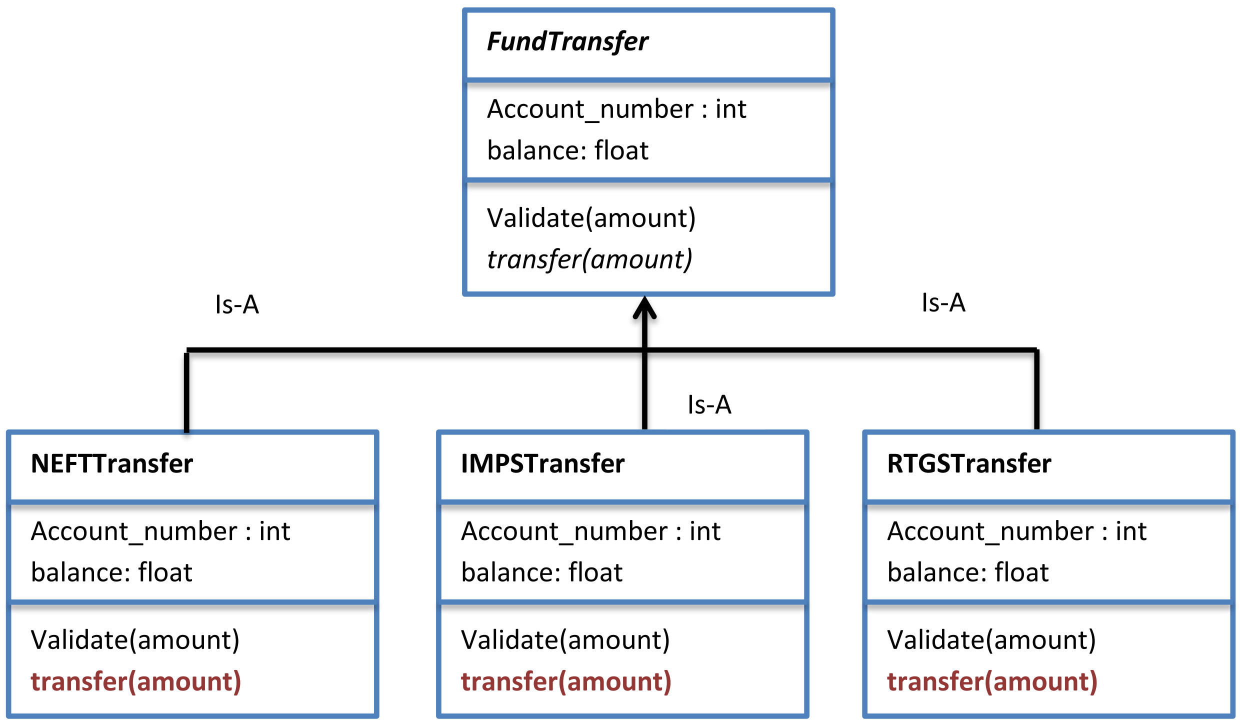
1. NEFT
2. IMPS
3. RTGS



1

Enter the amount to be transferred : 2000

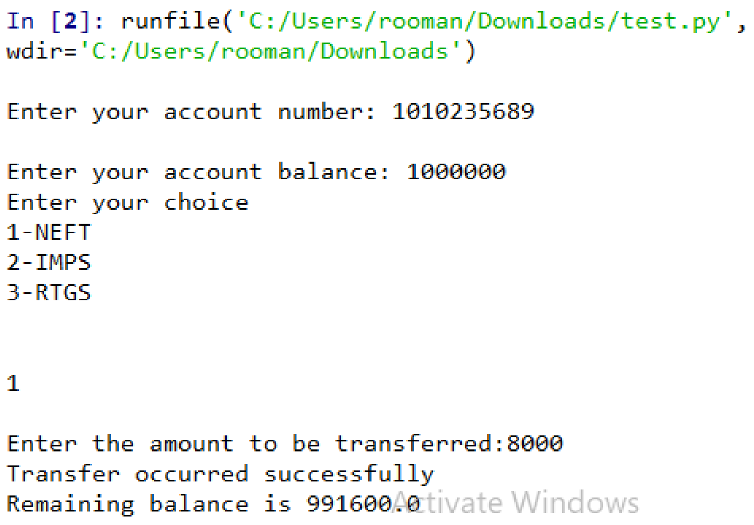
Transfer occurred successfully remaining balance is 7900.0



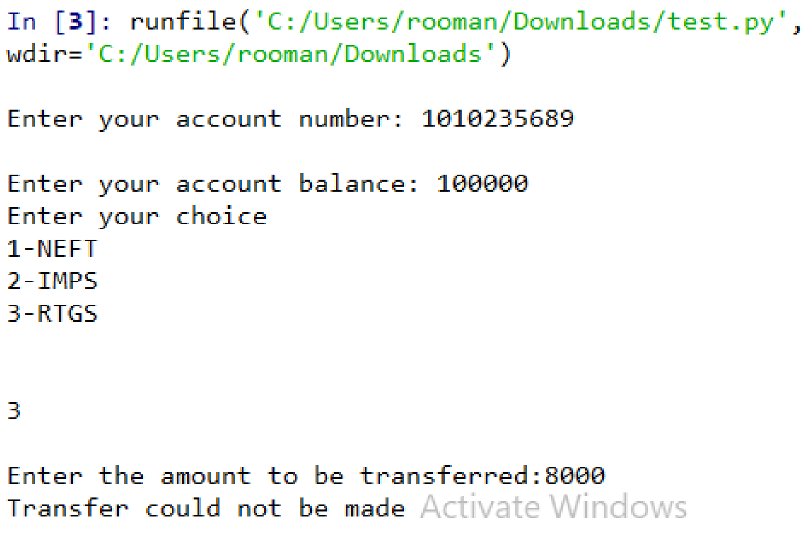
```python
from abc import ABC, abstractmethod

class FundTransfer(ABC):

    def __init__(self,account_number,balance):
        self.__account_number=account_number
        self.__balance=balance

    @property
    def account_number(self):
        return self.__account_number

    @account_number.setter
    def account_number(self,account_number):
        if len(str(account_number))==10:
            self.__account_number=account_number

    @property
    def balance(self):
        return self.__balance

    @balance.setter
    def balance(self,balance):
        if balance>0:
            self.__balance=balance
    def validate(self,amount):
        return len(str(self.account_number))==10 and\
        amount<self.__balance and amount>0

    @abstractmethod
    def transfer(self,amount):
        pass
```

```python
class NEFTTransfer(FundTransfer):

    def __init__(self,account_number,balance):
        super().__init__(account_number,balance)

    def transfer(self,amount):
        sc=amount*0.05
        if (amount+sc)<self.balance:
            self.balance=self.balance-(amount+sc)
            return True
        return False

class IMPSTransfer(FundTransfer):

    def __init__(self,account_number,balance):
        super().__init__(account_number,balance)

    def transfer(self,amount):
        sc=amount*0.02
        if (amount+sc)<self.balance:
            self.balance=self.balance-(amount+sc)
            return True
        return False

class RTGSTransfer(FundTransfer):

    def __init__(self,account_number,balance):
        super().__init__(account_number,balance)

    def transfer(self,amount):
        if amount<self.balance and amount>=10000:
            self.balance=self.balance-amount
            return True
        return False


def main():
    an=int(input("Enter your account number: "))
    bal=int(input("Enter your account balance: "))

    print("Enter your choice")
    print("1-NEFT\n2-IMPS\n3-RTGS\n")
    choice=int(input())

    if choice==1:
        ref=NEFTTransfer(an,bal)
    elif choice==2:
        ref=IMPSTransfer(an,bal)
    elif choice==3:
        ref=RTGSTransfer(an,bal)
    else:
        print('INVALID CHOICE')

    amt=int(input('Enter the amount to be transferred:'))

    if ref.validate(amt):
        if ref.transfer(amt):
            print('Transfer occurred successfully')
            print(f'Remaining balance is {ref.balance}')
        else:
            print('Transfer could not be made')
    else:
        print('Account number or transfer amount seems to be wrong')

if __name__=='__main__':
    main()
```

Output:



```
In [2]: runfile('C:/Users/rooman/Downloads/test.py',
wdir='C:/Users/rooman/Downloads')

Enter your account number: 1010235689

Enter your account balance: 1000000
Enter your choice
1-NEFT
2-IMPS
3-RTGS


1

Enter the amount to be transferred:8000
Transfer occurred successfully
Remaining balance is 991600.0
```



```
In [3]: runfile('C:/Users/rooman/Downloads/test.py',
wdir='C:/Users/rooman/Downloads')

Enter your account number: 1010235689

Enter your account balance: 100000
Enter your choice
1-NEFT
2-IMPS
3-RTGS


3

Enter the amount to be transferred:8000
Transfer could not be made
```

As RTGS expects minimum of Rs.10000 transfer could not be made.