Decorator

A   B   C

fb.com → Pretask   fb.com   fb.com
login   login   login
                          Liked   comment
Upload Photo   logout
                          logout
logout
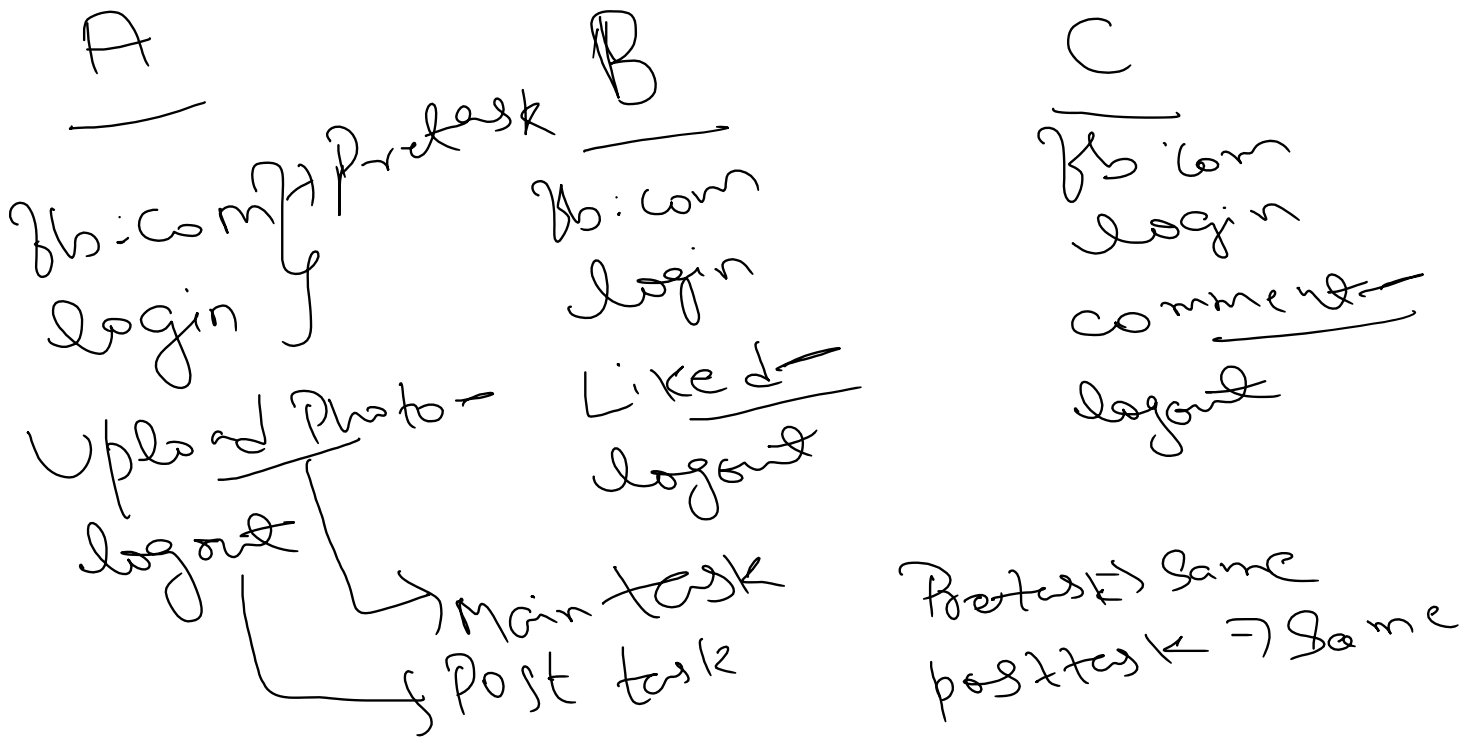   → Main task
                  Post task

Pretask → Same
posttask → Same

Decorator is a function which is used to add some extra features to the main task without modifying it.

Eg:
```
def A():
    print('fb.com')
    print('login')
    print('upload photo')
    print('logout')
A()
def B():
    print('fb.com')
    print('login')
    print('like')
    print('logout')
B()
def C():
    print('fb.com')
    print('login')
    print('comment')
    print('logout')
C()
```

Here there are 3 main drawbacks
   i. It will increase code length.
  ii. Increase code repeatation.
 iii. Decrease efficiency.

To avoid this issue, decorator is used.

There are 2 types of decorators'
   i. In-built decorator
  ii. User defined decorator

1. In-built decorator: These are the decorators that is already designed by the developers, we can use it but can't modify it.
   Eg:
      i. @staticmethod
     ii. @classmethod
    iii. @abstractmethod
    iv. @property

2. User-defined Decorator: It is the decorator that is created based upon the user requirement.

   Syntax:
   ```
   Def deco_name(func):
        Def inner(*args,**kwargs): #packing
   ```

```
        Pretask
        Func(*args,*kwargs) #unpacking
        Posttask
    Return inner
@deco_name
Def fname(args):
    SB
Fname(values)


Eg:
def fb(func):
    def inner(*args,**kwargs):
        print('fb.com')
        print('login')
        func(*args,*kwargs)
        print('logout')
    return inner
@fb
def A():
    print('upload photo')
A()
@fb
def B():
    print('like')
B()
@fb
def C():
    print('Comment')
C()
```
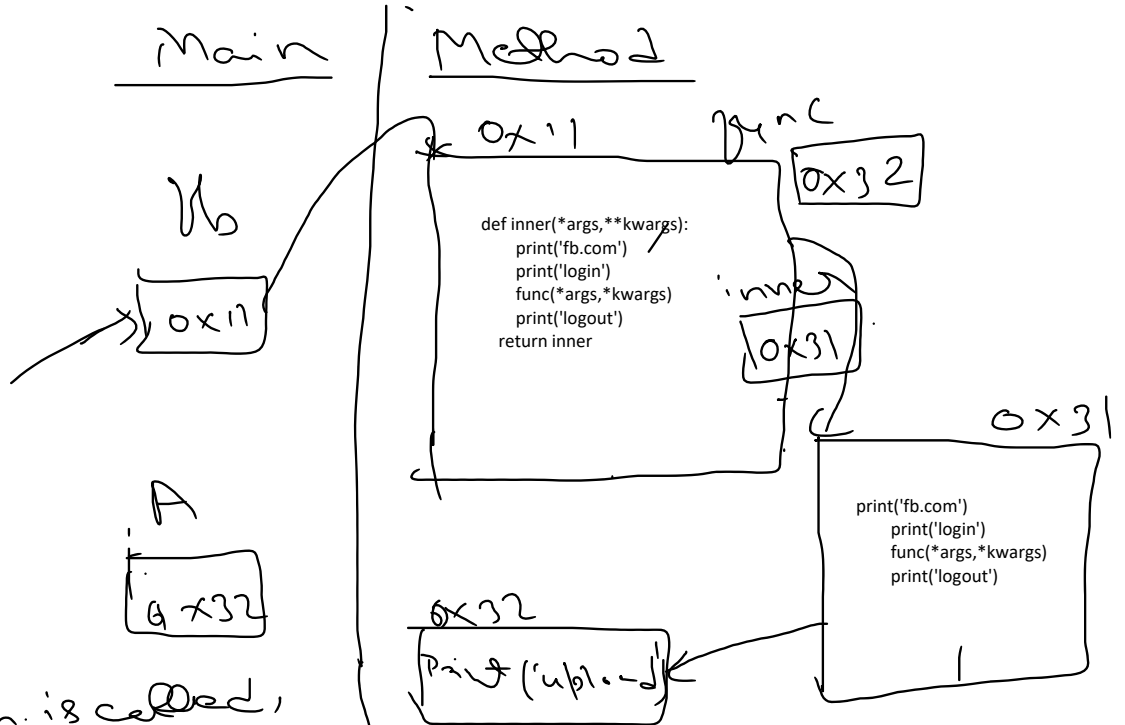

Main | Method

If main fn. is called, it actually calls inner() and main fn. internally gets executed.

fb.com upload login logout

```
Eg:
#WAP to create a decorator to provide 5 seconds delay before executing the program.
#import time
#time.sleep(5)
def delay(func):
    def inner(*args,**kwargs):
        import time
        time.sleep(5)
        func(*args,*kwargs)
    return inner
@delay
def sam():
    print('Hii this is python class')
sam()
```

Assign:
1. WAP to create a decorator that returns only positive values.
2. WAP to create a decorator which should tell the number of times a function is called.
3. WAP to create a decorator to wish happy diwali to all the employees in the company.