Видео-разбор подвига (решение смотреть только после своей попытки): <https://youtu.be/d5aNVdHGj44>

**Подвиг 7 (введение в паттерн слушатель).** Своей работой вы немного впечатлили начальство и оно поручило вам доделать паттерн слушатель (listener). Идея этого паттерна очень проста и основа реализуется следующим образом:

class Observer:

def update(self, data):

pass

def \_\_hash\_\_(self):

return hash(id(self))

class Subject:

def \_\_init\_\_(self):

self.\_\_observers = {}

self.\_\_data = None

def add\_observer(self, observer):

self.\_\_observers[observer] = observer

def remove\_observer(self, observer):

if observer in self.\_\_observers:

self.\_\_observers.pop(observer)

def \_\_notify\_observer(self):

for ob in self.\_\_observers:

ob.update(self.\_\_data)

def change\_data(self, data):

self.\_\_data = data

self.\_\_notify\_observer()

Здесь в объектах класса Subject можно зарегистрировать (добавить) множество объектов класса Observer (наблюдатель, слушатель). Это делается с помощью метода add\_observer(). Затем, когда данные (self.\_\_data) меняются путем вызова метода change\_data() класса Subject, то у всех слушателей автоматически вызывается метод update(). В этом методе можно прописать самую разную логику работы при изменении данных в каждом конкретном слушателе.

В проекте данный паттерн предполагается использовать для отображения информации о погоде в различных форматах:

- текущая температура;  
- текущее атмосферное давление;  
- текущая влажность воздуха.

Для этого сами данные определяются классом:

class Data:

    def \_\_init\_\_(self, temp, press, wet):

        self.temp = temp    # температура

        self.press = press  # давление

        self.wet = wet      # влажность

А вам поручается разработать дочерние классы, унаследованные от класса Observer, с именами:

**TemperatureView** - слушатель для отображения информации о температуре;  
**PressureView** - слушатель для отображения информации о давлении;  
**WetView** - слушатель для отображения информации о влажности.

Каждый из этих классов должен переопределять метод update() базового класса так, чтобы выводилась в консоль информация в формате:

TemperatureView: "Текущая температура <число>"  
PressureView: "Текущее давление <число>"  
WetView: "Текущая влажность <число>"

**Важно:** для вывода информации в консоль используйте функцию print() с одним аргументом в виде F-строки.

Пример использования классов (эти строчки в программе писать не нужно):

subject = Subject()

tv = TemperatureView()

pr = PressureView()

wet = WetView()

subject.add\_observer(tv)

subject.add\_observer(pr)

subject.add\_observer(wet)

subject.change\_data(Data(23, 150, 83))

# выведет строчки:

# Текущая температура 23

# Текущее давление 150

# Текущая влажность 83

subject.remove\_observer(wet)

subject.change\_data(Data(24, 148, 80))

# выведет строчки:

# Текущая температура 24

# Текущее давление 148

P.S. В программе нужно объявить только классы. На экран выводить ничего не нужно.

class Observer:

def update(self, data):

pass

def \_\_hash\_\_(self):

return hash(id(self))

class Subject:

def \_\_init\_\_(self):

self.\_\_observers = {}

self.\_\_data = None

def add\_observer(self, observer):

self.\_\_observers[observer] = observer

def remove\_observer(self, observer):

if observer in self.\_\_observers:

self.\_\_observers.pop(observer)

def \_\_notify\_observer(self):

for ob in self.\_\_observers:

ob.update(self.\_\_data)

def change\_data(self, data):

self.\_\_data = data

self.\_\_notify\_observer()

class Data:

def \_\_init\_\_(self, temp, press, wet):

self.temp = temp # температура

self.press = press # давление

self.wet = wet # влажность

# здесь объявляйте дочерние классы TemperatureView, PressureView и WetView