Видео-разбор подвига (решение смотреть только после своей попытки): <https://youtu.be/V1fqV9pfARQ>

**Подвиг 8 (на повторение).** В программе объявлен базовый класс Function (функция) следующим образом:

class Function:

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

self.\_amplitude = 1.0 # амплитуда функции

self.\_bias = 0.0 # смещение функции по оси Oy

def \_\_call\_\_(self, x, \*args, \*\*kwargs):

return self.\_amplitude \* self.\_get\_function(x) + self.\_bias

def \_get\_function(self, x):

raise NotImplementedError('метод \_get\_function должен быть переопределен в дочернем классе')

def \_\_add\_\_(self, other):

if type(other) not in (int, float):

raise TypeError('смещение должно быть числом')

obj = self.\_\_class\_\_(self)

obj.\_bias = self.\_bias + other

return obj

Здесь в инициализаторе создаются два локальных атрибута:

\_amplitude - амплитуда функции;  
\_bias - смещение функции по оси ординат (Oy).

Далее, в методе \_\_call\_\_() берется значение функции в точке x через метод \_get\_function(), который должен быть определен в дочерних классах, умножается на амплитуду функции и добавляется ее смещение. Следующий метод \_\_add\_\_() позволяет менять смещение функции, изменяя атрибут \_bias на указанное значение other.

Обратите внимание, в методе \_\_add\_\_() происходит создание нового объекта командой:

obj = self.\_\_class\_\_(self)

Здесь \_\_class\_\_ - это ссылка на класс, к которому относится объект self. Благодаря этому в базовом классе можно создавать объекты соответствующих дочерних классов. В момент создания объекта ему передается параметр self как аргумент. Так будет создаваться копия объекта, т.е. новый объект с тем же набором и значениями локальных атрибутов.

Чтобы обеспечить этот функционал, объявите дочерний класс с именем **Linear** (линейная функция y = k\*x + b), объекты которого должны создаваться командами:

obj = Linear(k, b)

linear = Linear(obj)  # этот вариант используется в базовом классе в методе \_\_add\_\_()

В первом случае происходит создание объекта линейной функции с параметрами k и b. Во втором - создание объекта со значениями параметров k и b, взятыми из объекта obj.

В каждом объекте класса Linear должны создаваться локальные атрибуты с именами \_k и \_b с соответствующими значениями.  
В результате будет создан универсальный базовый класс Function для работы с произвольными функциями от одного аргумента.

Применять эти классы можно следующим образом (эти строчки в программе писать не нужно):

f = Linear(1, 0.5)

f2 = f + 10   # изменение смещения (атрибут \_bias)

y1 = f(0)     # 0.5

y2 = f2(0)    # 10.5

Пропишите в базовом классе Function еще один магический метод для изменения масштаба (амплитуды) функции, чтобы был доступен оператор умножения:

f = Linear(1, 0.5)

f2 = f \* 5    # изменение амплитуды (атрибут \_amplitude)

y1 = f(0)     # 0.5

y2 = f2(0)    # 2.5

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

class Function:

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

self.\_amplitude = 1.0 # амплитуда функции

self.\_bias = 0.0 # смещение функции по оси Oy

def \_\_call\_\_(self, x, \*args, \*\*kwargs):

return self.\_amplitude \* self.\_get\_function(x) + self.\_bias

def \_get\_function(self, x):

raise NotImplementedError('метод \_get\_function должен быть переопределен в дочернем классе')

def \_\_add\_\_(self, other):

if type(other) not in (int, float):

raise TypeError('смещение должно быть числом')

obj = self.\_\_class\_\_(self)

obj.\_bias = self.\_bias + other

return obj

# здесь добавляйте еще один магический метод для умножения

# здесь объявляйте класс Linear