Треугольник

Написать класс для треугольника на плоскости **Triangle**. В классе всего 3 поля - длины сторон. Нужно написать следующие методы:

1. Конструктор. (**__init__(self, l1, l2, l3)**). Должен проверять, можно ли создать такой треугольник. Если нельзя - бросает эксепшн (материал об этом будет на будущей лекции).
2. Периметр (**perimeter(self)**). Считает и возвращает периметр треугольника.
3. Площадь (**area(self)**). Считает и возвращает площадь.
4. Равнобедренный ли (**isosceles(self)**). Возвращает True, если треугольник равнобедренный.
5. Равносторонний ли (**equilateral(self)**). Возвращает True, если треугольник равносторонний.
6. Равен ли другому треугольнику. (Перегрузить проверку на равенство)

Окружность

Написать класс для окружности на плоскости **Circle**. У класса два поля - координаты и радиус. Координаты задаются массивом из двух чисел. Нужно написать методы:

1. Конструктор со значениями по умолчанию (**__init__(self, coords, r)**)
2. Площадь (**area(self)**). Считает и возвращает площадь.
3. Периметр (**perimeter(self)**). Считает и возвращает периметр.
4. Переместить круг в конкретную точку (**move_to_point(self, coords)**)
5. Изменить радиус (**set_radius(self, r)**)
6. Определить, пересекается ли окружность с другой (перегрузить &)
7. Определить, находится окружность внутри другой полностью (перегрузить <, >, =)

Товар

Написать класс **Item** для различных товаров, которые могут быть в магазине (нужно для следующего задания). Например компьютер, экран, мышка, флешка, телефон. Атрибуты передаются в конструкторе.

Поля:

- Name (название товара)
- Price (цена товара)
- Category (категория товара (Смартфон, Носки, Ноутбук))

Магазин товаров

Написать класс магазина **Shop**. Методы:

- Добавить новый товар (**add(self, item))**
- Узнать количество товаров определенного вида (**count_category(self, category)**). Возвращает количество товаров.
- Искать товары по параметрам: диапазон цены, список категорий (**find_items(self, price_bounds, categories)**). Возвращает список товаров.

price_bounds - список длины 2.
Левый, правый, или оба конца **price_bounds** могут быть не заданы (передано None), это значит, что искать нужно подставлять -inf, inf или [-inf, inf] соответственно.
categories - список категорий.
categories может быть не задано (None) - это значит, искать нужно по всем категориям.
- Узнать общую стоимость товаров по видам и общую стоимость всех товаров (**get_full_price(self, categories)**). Возвращает стоимость.

categories - список категорий.
categories может быть не задано - это значит, искать нужно по всем категориям.
- Узнать наличие конкретного товара (**is_available(item)**). Возвращает True/False.

item - объект класса Item
Обязательно перегрузить in или равенство для товаров (в зависимости от реализации класса **Shop**).

Роботы

Написать класс робота **Robot**, который может драться с другим роботом. За это будет отвечать метод robot1.fightWith(robot2).

Обязательные атрибуты:

- name - имя робота
- power - максимальная сила удара робота
- health - текущий уровень здоровья

Обязательные методы:
- __init__(self, name=None, power=10) - в этом методе происходит инициализация обязательных атрибутов. Если name не передано, то имя робота должно генерироваться *автоматически* п о правилу: “Robot id”, где id - уникальный идентификатор робота (например, порядковый номер объекта класса). Пользоваться global нельзя.
По-умолчанию на старте у каждого робота 100 единиц здоровья.
- hit(self, robot) - в этом методе происходит удар по роботу robot. Наносимый урон - случайное целое число от 1 до power того робота, который бьет. Возвращает урон, который нанес робот.
- fightWith(self, robot) - драка двух роботов. Первый бьет тот, кто начал драку :) Возвращает True, если выиграл первый, и False в противном случае. При этом у проигравшего должно оставаться 0 здоровья. Для битвы нужно написать код так, чтобы они по очереди наносили случайный урон друг другу (каждый удар

выводится в виде текста вместе с информацией о здоровье).

Пример:

Робот Robot 1 ударил робота Robot 2 и нанес 4 урона здоровью Здоровье робота Robot 1: 100, здоровье робота Robot 2: 96
Робот Robot 2 ударил робота Robot 1 и нанес 68 урона здоровью Здоровье робота Robot 2: 96, здоровье робота Robot 1: 32
Робот Robot 1 ударил робота Robot 2 и нанес 2 урона здоровью Здоровье робота Robot 1: 32, здоровье робота Robot 2: 94
Робот Robot 2 ударил робота Robot 1 и нанес 52 урона здоровью Здоровье робота Robot 2: 94, здоровье робота Robot 1: 0
Робот Robot 2 побеждает!

Вывод информации о каждом ударе должен быть реализован с помощью **декоратора**. Вывод информации о победителе можно реализовывать без помощи декоратора.

Поиск пути в лабиринте

Написать программу поиска в ШИРИНУ кратчайшего пути в лабиринте 10 на 10 клеток между заданными точками старта и финиша. Ходить можно по горизонтали или вертикали. Алгоритм поиска нужно написать самому. Нельзя пользоваться numpy. Нельзя пользоваться другими готовыми решениями.

Предполагается что вы представите поле в виде графа, каждая его клетка - узел (объект класса Node). В узле хранятся ссылки на соседние клетки, тип узла (поле, стена, старт, финиш), минимальное расстояние от старта и знание о том, какой узел является предыдущем в кратчайшем пути. Последние два поля заполняются при нахождении кратчайшего пути.

Само поле задается с помощью класса Board. У класса должны быть методы:

read - считать поле, задается строкой в формате указанном ниже (горизонтали разделены с помощью \n)

solve - найти кратчайший путь и сохранить его (сам путь и его длину)

print - вывести считанный лабиринт с кратчайшим путем (можно сделать опцию просто вывести лабиринт до вызова метода solve)

Для отладки вам может быть полезно сделать метод generate, который создает случайное поле.

В объекте класса Board можно хранить только два объекта класса Node - левый верхний угол и точка старта (S).

Вход:

рисунок лабиринта

0X0X000000
00000X0000
0000000000
00000F0000
0000000000
0000000000
XX0XX00000
0000X00000
XXX0X00000
S000X00000

Обозначения: O = пустое поле, X - стена, S - старт, F - финиш

Выход:

Длина кратчайшего пути между стартом и финишем (Включая конечные точки). Если путь существует, то вывести так же лабиринт, в котором цветом выделить кратчайший путь.

14

0X0X000000
00000X0000
0000000000
00000F0000
0000000000
0000000000
XX0XX00000
0000X00000
XXX0X00000
S000X00000

Для того, чтобы написать текст в консоли каким-то цветом, можно использовать ansi-код этого цвета:

RED = '\033[31m'

```
END = '\033[0m'

print(RED + ‘red text’ + END)
```