1. Весы

Напишите класс **Balance** для описания весов с двумя чашами. На левую и правую чашу объекта будут добавляться грузы с различным весом, ваша задача определить положение чаш.

Метод add\_right принимает целое число — вес, положенный на правую чашу весов, add\_left — на левую чашу. Метод result должен возвращать символ =, если вес на чашах одинаковый, R — если перевесила правая, L — если перевесила левая.

Пример 1

| Ввод | Вывод |
| --- | --- |
| balance = Balance()  balance.add\_right(10)  balance.add\_left(9)  balance.add\_left(2)  print(balance.result()) | L |

Пример 2

| Ввод | Вывод |
| --- | --- |
| balance = Balance()  balance.add\_right(10)  balance.add\_left(5)  balance.add\_left(5)  print(balance.result())  balance.add\_left(1)  print(balance.result()) | =  L |

1. Вектор на плоскости

Напишите класс MyVector, описывающий геометрический вектор и операции с ним.

Операции: сложение, вычитание, умножение на константу (в случае, когда константа стоит слева и справа, умножение \*=), вывод , сравнение на равенство и неравенство, длина вектора (\_\_abs\_\_). Данные операции необходимо реализовать через **специальные функции**.

Пример 1

| Ввод | Вывод |
| --- | --- |
| v1 = MyVector(-2, 5)  v2 = MyVector(3, -4)  v\_sum = v1 + v2  print(v\_sum) | MyVector(1, 1) |

1. Сложение многочленов

Реализуйте класс Polynomial для вычисления многочлена при заданном х, а также для сложения многочленов.

Конструктор класса Polynomial(coefficients) принимает список из коэффициентов.

Вызов экземпляра класса (необходимо использовать соответствующий специальный метод) вычисляет значение в переданной точке.

В результате сложения двух многочленов должен создаваться новый экземпляр класса Polynomial.

**Пример 1**

| Ввод | Вывод |
| --- | --- |
| poly = Polynomial([10, -1])  print(poly(0))  print(poly(1))  print(poly(2)) | 10  9  8 |

### Пример 2

| Ввод | Вывод |
| --- | --- |
| poly1 = Polynomial([0, 1])  poly2 = Polynomial([10])  poly3 = poly1 + poly2  poly4 = poly2 + poly1  print(poly3(-2), poly4(-2))  print(poly3(-1), poly4(-1))  print(poly3(0), poly4(0))  print(poly3(1), poly4(1))  print(poly3(2), poly4(2)) | 8 8  9 9  10 10  11 11  12 12 |

1. **СМС-рассылка**

Для создания смс-рассылки напишите классы **Person** и **Company**, а также функцию **send\_sms**.

Объект класса **Person** при инициализации принимает значения имени, отчества и фамилии человека, а также словарь с номерами его телефонов.

Класс должен содержать следующие методы:

1. **get\_phone()** – возвращает телефон из словаря по ключу 'private' или None, если такого телефона нет,
2. **get\_name()** – возвращает фамилию, имя и отчество человека через пробел,
3. **get\_work\_phone()** – возвращает телефон из словаря по ключу 'work' или None, если такого телефона нет,
4. **get\_sms\_text()** – возвращает текст «Уважаемый <имя> <отчество>! Примите участие в нашем беспроигрышном конкурсе для физических лиц».

Объект класса **Company** при инициализации принимает название компании, тип компании, словарь с её телефонами, а также неограниченное количество работников компании (объектов класса **Person**).

Класс должен содержать следующие методы:

1. **get\_phone()** – возвращает телефон из словаря по ключу 'contact', если его нет, то телефон первого работника, у которого есть телефон по ключу 'work', или None, если таких работников не найдётся,
2. **get\_name()** – возвращает название компании,
3. **get\_sms\_text()** – возвращает текст «Для компании <название компании> есть супер предложение! Примите участие в нашем беспроигрышном конкурсе для <тип компании>».

Функция **send\_sms** должна принимать неограниченное количество объектов класса **Person** или **Company** и в случае, если найден номер для отправки (с помощью метода get\_phone()), выводить сообщение «Отправлено СМС на номер <номер> с текстом: <Текст СМС>», иначе – текст «Не удалось отправить сообщение абоненту: <ФИО человека или название компании>».

**Пример 1**

| Ввод | Вывод |
| --- | --- |
| person1 = Person("Ivan", "Ivanovich", "Ivanov", {"private": 123, "work": 456})  person2 = Person("Ivan", "Petrovich", "Petrov", {"private": 789})  person3 = Person("Ivan", "Petrovich", "Sidorov", {"work": 789})  person4 = Person("John", "Unknown", "Doe", {})  company1 = Company("Bell", "ООО", {"contact": 111}, person3, person4)  company2 = Company("Cell", "АО", {"non\_contact": 222}, person2, person3)  company3 = Company("Dell", "Ltd", {"non\_contact": 333}, person2, person4)  send\_sms(person1, person2, person3, person4, company1, company2, company3) | Отправлено СМС на номер 123 с текстом: Уважаемый Ivan Ivanovich! Примите участие в нашем беспроигрышном конкурсе для физических лиц  Отправлено СМС на номер 789 с текстом: Уважаемый Ivan Petrovich! Примите участие в нашем беспроигрышном конкурсе для физических лиц  Не удалось отправить сообщение абоненту: Sidorov Ivan Petrovich  Не удалось отправить сообщение абоненту: Doe John Unknown  Отправлено СМС на номер 111 с текстом: Для компании Bell есть супер предложение! Примите участие в нашем беспроигрышном конкурсе для ООО  Отправлено СМС на номер 789 с текстом: Для компании Cell есть супер предложение! Примите участие в нашем беспроигрышном конкурсе для АО  Не удалось отправить сообщение абоненту: Dell |

**Пример 2**

| Ввод | Вывод |
| --- | --- |
| person1 = Person("Степан", "Петрович", "Джобсов", {"private": 555})  person2 = Person("Боря", "Иванович", "Гейтсов", {"private": 777, "work": 888})  person3 = Person("Семен", "Робертович", "Возняцкий", {"work": 789})  person4 = Person("Леонид", "Арсенович", "Торвальдсон", {})  company1 = Company("Яблочный комбинат", "ООО", {"contact": 111}, person1, person3)  company2 = Company("ПластОкно", "АО", {"non\_contact": 222}, person2)  company3 = Company("Пингвинья ферма", "Ltd", {"non\_contact": 333}, person4)  send\_sms(person1, person2, person3, person4, company1, company2, company3) | Отправлено СМС на номер 555 с текстом: Уважаемый Степан Петрович! Примите участие в нашем беспроигрышном конкурсе для физических лиц  Отправлено СМС на номер 777 с текстом: Уважаемый Боря Иванович! Примите участие в нашем беспроигрышном конкурсе для физических лиц  Не удалось отправить сообщение абоненту: Возняцкий Семен Робертович  Не удалось отправить сообщение абоненту: Торвальдсон Леонид Арсенович  Отправлено СМС на номер 111 с текстом: Для компании Яблочный комбинат есть супер предложение! Примите участие в нашем беспроигрышном конкурсе для ООО  Отправлено СМС на номер 888 с текстом: Для компании ПластОкно есть супер предложение! Примите участие в нашем беспроигрышном конкурсе для АО  Не удалось отправить сообщение абоненту: Пингвинья ферма |

E. Очередь

Очередь – это такая структура данных, работа с которой определяется принципом FIFO (First In First Out). **Очередь может быть инициализирована** различным числом элементов.

Например,

q = Queue(1, 2, 3, 4, 5) – создаёт очередь [1 -> 2 -> 3 -> 4 -> 5], а

q1 = Queue(1, 2, 3) – очередь [1 -> 2 -> 3].

Но всё же есть гарантия того, что для инициализации передается хотя бы один параметр.

**Класс Queue реализует следующие методы:**

**append(∗values)** – добавляет несколько значений в конец очереди (как минимум одно).

**copy()** – создаёт копию данной очереди, то есть возвращает новую очередь, полностью аналогичную исходной.

**pop()** – вытаскивает и возвращает первый элемент очереди, при этом этот элемент из очереди удаляется. Если очередь пуста, то возвращает None.

**extend(queue)** – расширяет данную очередь другой, то есть приклеивает вторую очередь к первой.

**next()** – возвращает новую очередь, начинающуюся со второго элемента текущей.

**Также требуется реализовать следующие операторы и встроенные** **функции:**

**queue1+queue2** – склейка очередей создаёт новую увеличенную очередь.

**queue1+=queue2** – расширяет первую очередь второй.

**queue1==queue2** – проверяет очереди на равенство всех элементов. Возвращает True или False.

**queue>>N** – создаёт новую очередь без первых N (вышедших) элементов. В случае, когда N превышает количество элементов очереди, следует вернуть пустую очередь.

**str(queue)** – приводит очередь к строке вида [q1 -> q2 -> q3 -> ... -> qn]. Пустая или ошибочная очередь – это []

**next(queue)** – аналогичное действие методу next()

## Пример

| Ввод | Вывод |
| --- | --- |
| q1 = Queue(1, 2, 3)  print(q1)  q1.append(4, 5)  print(q1)  qx = q1.copy()  print(qx.pop())  print(qx)  q2 = q1.copy()  print(q2)  print(q1 == q2, id(q1) == id(q2))  q3 = q2.next()  print(q1, q2, q3, sep = '\n')  print(q1 + q3)  q3.extend(Queue(1, 2))  print(q3)  q4 = Queue(1, 2)  q4 += q3 >> 4  print(q4)  q5 = next(q4)  print(q4)  print(q5) | [1 -> 2 -> 3]  [1 -> 2 -> 3 -> 4 -> 5]  1  [2 -> 3 -> 4 -> 5]  [1 -> 2 -> 3 -> 4 -> 5]  True False  [1 -> 2 -> 3 -> 4 -> 5]  [1 -> 2 -> 3 -> 4 -> 5]  [2 -> 3 -> 4 -> 5]  [1 -> 2 -> 3 -> 4 -> 5 -> 2 -> 3 -> 4 -> 5]  [2 -> 3 -> 4 -> 5 -> 1 -> 2]  [1 -> 2 -> 1 -> 2]  [1 -> 2 -> 1 -> 2]  [2 -> 1 -> 2] |

**F. Сумматоры**

Реализуйте несколько классов, вычисляющих суммы следующих последовательностей натуральных чисел от 1 до N:

* 
* 
* 

Сделайте это с помощью наследования.  
Базовым классом будет Summator, который реализует методы transform(n) и sum(N).

Метод transform(n) выполняет преобразование элемента последовательности. Для класса Summator это будет тождественное преобразование, т.е. transform(n) == n. Метод transform(n) будет переопределяться в производных классах.

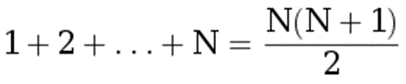
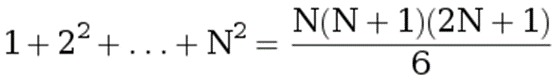
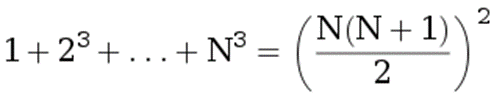
Метод sum(N) должен вычислять сумму значений transform(n) для всех значений n от 1 до N.

Производные классы должны переопределять метод transform(n):

* SquareSummator: transform(n) == n \*\* 2
* CubeSummator: transform(n) == n \*\* 3

Метод sum(N) переопределять не нужно.

**Подсказка:** для проверки корректности реализации классов Summator, SquareSummator, CubeSummator воспользуйтесь формулами:

* 
* 
* 

**G. Заготовка для игры**

При разработке компьтерных игр зачастую создаётся очень сложная система классов. В этой задаче необходимо сделать небольшую заготовку, которую потом можно будет использовать для создания своей игры.

Реализуйте класс **Weapon** для оружия, который при инициализации должен принимать имя оружия **name**, наносимый урон **damage**, а также радиус, на котором оружие может достать до врага – **range**.

В классе **Weapon** должны быть реализованы следующие методы:

1. **hit(actor, target)** – удар персонажем actor персонажа target. Метод должен проверять, жив ли персонаж target, если нет, то выводить сообщение «Враг уже повержен», иначе должна проводится проверка расстояния от actor до target. Если расстояние больше range оружия, тогда надо выводить сообщение «Враг слишком далеко для оружия <name>». В случае, если target жив и оружие имеет достаточный range, вывести сообщение «Врагу нанесен урон оружием <name> в размере <damage>» и нанести урон target.
2. приведение к строке – при приведении к строке должно возвращаться имя оружия. Реализуйте класс **BaseCharacter** для создания базовых персонажей, который при инициализации должен принимать **pos\_x** и **pos\_y** – позицию персонажа в мире, **hp** – количество жизней персонажа.

Класс **BaseCharacter** должен реализовывать методы:

1. **move(delta\_x, delta\_y)** – для перемещения персонажа на delta\_x и delta\_y в игровом мире.
2. **is\_alive()** – проверка, жив ли персонаж. Возвращает True, если жив, иначе – False.
3. **get\_damage(amount)** – убавляет количество жизней персонажа на amount. При достижении 0 или отрицательного значения персонаж считается мёртвым.
4. **get\_coords()** – возвращает кортеж с текущими координатами персонажа.

Реализуйте класс **BaseEnemy** для создания противников, который расширяет класс

BaseCharacter. При инициализации он должен принимать **pos\_x** и **pos\_y** – позицию персонажа в мире, **weapon** – оружие, **hp** – количество жизней персонажа. Класс должен использовать инициализатор BaseCharacter.

Класс **BaseEnemy** должен реализовывать следующие методы:

1. **hit(target)** – для удара персонажа target. Метод должен позволять ударить только персонажа класса MainHero. Для всех других классов должно печататься сообщение «Могу ударить только Главного героя».
2. приведение к строке – при приведении к строке должно возвращаться «Враг на позиции (<pos\_x>, <pos\_y>) с оружием <weapon>»

Реализуйте класс **MainHero** для создания главного героя, который расширяет класс BaseCharacter. При инициализации он должен принимать **pos\_x** и **pos\_y** – позицию персонажа в мире, **name** – имя персонажа, **hp** – количество жизней персонажа. Класс должен использовать инициализатор BaseCharacter.

Класс MainHero должен реализовывать следующие методы:

1. **hit(target)** – для удара персонажа target. Метод должен проверять, есть ли у персонажа оружие, если его нет выводить - «Я безоружен», иначе позволять ударить только персонажа класса BaseEnemy. Для всех других классов должно печататься сообщение «Могу ударить только Врага».
2. **add\_weapon(weapon)** – добавляет оружие в инвентарь персонажа, если это оружие единственное, то оно должно экипироваться сразу же. Метод должен печатать: «Подобрал <weapon>». Метод должен работать только для переданных объектов класса Weapon, для всего остального он должен печатать сообщение «Это не оружие».
3. **next\_weapon()** – для смены оружия. Если оружия нет, то должно выводиться « безоружен», если есть только одно оружие – «У меня только одно оружие», иначе должна производиться смена оружия на следующее (в порядке подбора, по кругу) и выводиться сообщение «Сменил оружие на <weapon>».
4. **heal(amount)** – для повышения количества жизней персонажа на amount. При этом максимальный уровень здоровья главного героя – 200. Также должно выводиться сообщение «Полечился, теперь здоровья <hp>».

## Пример

| Ввод | Вывод |
| --- | --- |
| weapon1 = Weapon("Короткий меч", 5, 1)  weapon2 = Weapon("Длинный меч", 7, 2)  weapon3 = Weapon("Лук", 3, 10)  weapon4 = Weapon("Лазерная орбитальная пушка", 1000, 1000)  princess = BaseCharacter(100, 100, 100)  archer = BaseEnemy(50, 50, weapon3, 100)  armored\_swordsman = BaseEnemy(10, 10, weapon2, 500)  archer.hit(armored\_swordsman)  armored\_swordsman.move(10, 10)  print(armored\_swordsman.get\_coords())  main\_hero = MainHero(0, 0, "Король Артур", 200)  main\_hero.hit(armored\_swordsman)  main\_hero.next\_weapon()  main\_hero.add\_weapon(weapon1)  main\_hero.hit(armored\_swordsman)  main\_hero.add\_weapon(weapon4)  main\_hero.hit(armored\_swordsman)  main\_hero.next\_weapon()  main\_hero.hit(princess)  main\_hero.hit(armored\_swordsman)  main\_hero.hit(armored\_swordsman) | Могу ударить только Главного героя (20, 20)  Я безоружен  Я безоружен  Подобрал Короткий меч  Враг слишком далеко для оружия Короткий меч  Подобрал Лазерная орбитальная пушка  Враг слишком далеко для оружия Короткий меч  Сменил оружие на Лазерная орбитальная пушка  Могу ударить только Врага  Врагу нанесен урон оружием Лазерная орбитальная пушка в размере 1000  Враг уже повержен |