# Введение в ООП

# Маленький колокольчик

Kлассная работа

макс. 1 балл.

|  |  |
| --- | --- |
| Ограничение времени | 1 секунда |
| Ограничение памяти | 64Mb |
| Ввод | стандартный ввод или testmodule.py |
| Вывод | стандартный вывод или output.txt |

Напишите класс **LittleBell**, который при вызове метода sound печатает слово "ding".

## Формат ввода

Каждый тест представляет собой код, в котором будет использоваться ваш класс. Файл c решением не обязательно называть solution.py, он будет переименован автоматически. Тест запускается с вашим классом, а его вывод сравнивается с правильным решением.

### Пример 1

| Ввод | Вывод |
| --- | --- |
| from solution import LittleBell  bell = LittleBell()  bell.sound() | ding |

### Пример 2

| Ввод | Вывод |
| --- | --- |
| from solution import LittleBell  bell = LittleBell()  bell.sound()  bell.sound()  bell.sound() | ding  ding  ding |

# Кнопка

Kлассная работа

макс. 1 балл.

|  |  |
| --- | --- |
| Ограничение времени | 1 секунда |
| Ограничение памяти | 64Mb |
| Ввод | стандартный ввод или testmodule.py |
| Вывод | стандартный вывод или output.txt |

Напишите класс кнопки **Button**, экземпляры которого будут измерять количество нажатий на кнопку-объект.

Метод click увеличивает количество нажатий, метод click\_count возвращает число нажатий. Метод reset обнуляет количество нажатий.

## Формат ввода

Каждый тест представляет собой код, в котором будет использоваться ваш класс. Файл c решением не обязательно называть solution.py, он будет переименован автоматически. Тест запускается с вашим классом, а его вывод сравнивается с правильным решением.

### Пример 1

| Ввод | Вывод |
| --- | --- |
| from solution import Button  button = Button()  button.click()  button.click()  print(button.click\_count())  button.click()  print(button.click\_count()) | 2  3 |

### Пример 2

| Ввод | Вывод |
| --- | --- |
| from solution import Button  button = Button()  button.click()  button.click()  print(button.click\_count())  button.reset()  button.click()  print(button.click\_count()) | 2  1 |

### Пример 3

| Ввод | Вывод |
| --- | --- |
| from solution import Button  button = Button()  button.click()  print(button.click\_count()) | 1 |

# Весы

Kлассная работа

макс. 2 балл.

|  |  |
| --- | --- |
| Ограничение времени | 1 секунда |
| Ограничение памяти | 64Mb |
| Ввод | стандартный ввод или testmodule.py |
| Вывод | стандартный вывод или output.txt |

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.

## Формат ввода

Каждый тест представляет собой код, в котором будет использоваться ваш класс. Файл c решением не обязательно называть solution.py, он будет переименован автоматически. Тест запускается с вашим классом, а его вывод сравнивается с правильным решением.

### Пример 1

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

### Пример 2

| Ввод | Вывод |
| --- | --- |
| from solution import Balance  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 |

# Разбивка по чётности

Kлассная работа

макс. 2 балл.

|  |  |
| --- | --- |
| Ограничение времени | 1 секунда |
| Ограничение памяти | 64Mb |
| Ввод | стандартный ввод или testmodule.py |
| Вывод | стандартный вывод или output.txt |

Напишите класс **OddEvenSeparator**, в который можно добавлять числа, получая потом отдельно чётные и нечётные. Числа добавляются в объект с помощью метода add\_number. Методы even и odd должны возвращать списки чётных и нечётных чисел соответственно. Числа в списке должны идти в том же порядке, что и при добавлении в объект.

## Формат ввода

Каждый тест представляет собой код, в котором будет использоваться ваш класс. Файл c решением не обязательно называть solution.py, он будет переименован автоматически. Тест запускается с вашим классом, а его вывод сравнивается с правильным решением.

## Пример

| Ввод | Вывод |
| --- | --- |
| from solution import OddEvenSeparator  separator = OddEvenSeparator()  separator.add\_number(1)  separator.add\_number(5)  separator.add\_number(6)  separator.add\_number(8)  separator.add\_number(3)  print(' '.join(map(str, separator.even())))  print(' '.join(map(str, separator.odd()))) | 6 8  1 5 3 |

# Большой колокольчик

Kлассная работа

макс. 2 балл.

|  |  |
| --- | --- |
| Ограничение времени | 1 секунда |
| Ограничение памяти | 64Mb |
| Ввод | стандартный ввод или testmodule.py |
| Вывод | стандартный вывод или output.txt |

Напишите класс **BigBell**, который при вызове метода sound печатает попеременно слова ding и dong, начиная c ding.

## Формат ввода

Каждый тест представляет собой код, в котором будет использоваться ваш класс. Файл c решением не обязательно называть solution.py, он будет переименован автоматически. Тест запускается с вашим классом, а его вывод сравнивается с правильным решением.

## Пример

| Ввод | Вывод |
| --- | --- |
| from solution import BigBell  bell = BigBell()  bell.sound()  bell.sound()  bell.sound() | ding  dong  ding |

# Самые короткие и самые длинные слова

Kлассная работа

макс. 2 балл.

|  |  |
| --- | --- |
| Ограничение времени | 1 секунда |
| Ограничение памяти | 64Mb |
| Ввод | стандартный ввод или testmodule.py |
| Вывод | стандартный вывод или output.txt |

Напишите класс **MinMaxWordFinder**. Класс должен уметь анализировать текст и находить в нём слова наименьшей и наибольшей длины. Текст состоит из предложений, которые добавляются в обработку методом add\_sentence. Метод shortest\_words возвращает список самых коротких на данный момент слов, метод longest\_words — самых длинных. Слова, возвращаемые методами shortest\_words и longest\_words, должны быть отсортированы по алфавиту.

Если одно из самых коротких слов встретилось в исходных предложениях несколько раз, оно должно столько же раз повториться в списке самых коротких слов. Самые длинные слова наоборот должны входить в список без повторов.

## Формат ввода

Каждый тест представляет собой код, в котором будет использоваться ваш класс. Файл c решением не обязательно называть solution.py, он будет переименован автоматически. Тест запускается с вашим классом, а его вывод сравнивается с правильным решением.

### Пример 1

| Ввод | Вывод |
| --- | --- |
| from solution import MinMaxWordFinder  finder = MinMaxWordFinder()  finder.add\_sentence('hello abc world')  finder.add\_sentence('def asdf qwert')  print(' '.join(finder.shortest\_words()))  print(' '.join(finder.longest\_words())) | abc def  hello qwert world |

### Пример 2

| Ввод | Вывод |
| --- | --- |
| from solution import MinMaxWordFinder  finder = MinMaxWordFinder()  finder.add\_sentence('hello')  finder.add\_sentence('abc')  finder.add\_sentence('world')  finder.add\_sentence('def')  finder.add\_sentence('asdf')  finder.add\_sentence('qwert')  print(' '.join(finder.shortest\_words()))  print(' '.join(finder.longest\_words())) | abc def  hello qwert world |

### Пример 3

| Ввод | Вывод |
| --- | --- |
| from solution import MinMaxWordFinder  finder = MinMaxWordFinder()  finder.add\_sentence('hello')  finder.add\_sentence(' abc def ')  finder.add\_sentence('world')  finder.add\_sentence(' abc ')  finder.add\_sentence('asdf')  finder.add\_sentence('qwert')  print(' '.join(finder.shortest\_words()))  print(' '.join(finder.longest\_words())) | abc abc def  hello qwert world |

# Ограничивающий прямоугольник

Kлассная работа

макс. 2 балл.

|  |  |
| --- | --- |
| Ограничение времени | 1 секунда |
| Ограничение памяти | 64Mb |
| Ввод | стандартный ввод или testmodule.py |
| Вывод | стандартный вывод или output.txt |

Создайте класс **BoundingRectangle**, который обрабатывает точки на плоскости и строит по ним прямоугольник минимального размера, в который входят все эти точки. Если точка лежит на границе прямоугольника, считается, что она в него входит.

Нужно определить следующие методы (rect – экземпляр **BoundingRectangle**):

rect.add\_point(x, y) — добавить новую точку.

rect.width() — ширина прямоугольника.

rect.height() — высота прямоугольника.

rect.bottom\_y() — Y-координата нижней границы прямоугольника.

rect.top\_y() — Y-координата верхней границы прямоугольника.

rect.left\_x() — X-координата левой границы прямоугольника.

rect.right\_x() — X-координата правой границы прямоугольника.

Гарантируется, что хотя бы одна точка будет добавлена в экземпляр до вызова методов, возвращающих описание прямоугольника.

## Формат ввода

Каждый тест представляет собой код, в котором будет использоваться ваш класс. Файл c решением не обязательно называть solution.py, он будет переименован автоматически. Тест запускается с вашим классом, а его вывод сравнивается с правильным решением.

### Пример 1

| Ввод | Вывод |
| --- | --- |
| from solution import BoundingRectangle  rect = BoundingRectangle()  rect.add\_point(-1, -2)  rect.add\_point(3, 4)  print(rect.left\_x(), rect.right\_x())  print(rect.bottom\_y(), rect.top\_y())  print(rect.width(), rect.height()) | -1 3  -2 4  4 6 |

### Пример 2

| Ввод | Вывод |
| --- | --- |
| from solution import BoundingRectangle  rect = BoundingRectangle()  rect.add\_point(10, 20)  rect.add\_point(5, 7)  rect.add\_point(6, 3)  print(rect.left\_x(), rect.right\_x())  print(rect.bottom\_y(), rect.top\_y())  print(rect.width(), rect.height()) | 5 10  3 20  5 17 |

### Пример 3

| Ввод | Вывод |
| --- | --- |
| from solution import BoundingRectangle  rect = BoundingRectangle()  rect.add\_point(-11, -12)  rect.add\_point(13, -14)  rect.add\_point(-15, 10)  print(rect.left\_x(), rect.right\_x())  print(rect.bottom\_y(), rect.top\_y())  print(rect.width(), rect.height())  print()  rect.add\_point(-21, -12)  rect.add\_point(13, -14)  rect.add\_point(-15, 36)  print(rect.width(), rect.height())  print(rect.left\_x(), rect.right\_x())  print(rect.bottom\_y(), rect.top\_y())  print()  rect.add\_point(-21, 78)  rect.add\_point(13, -14)  rect.add\_point(-55, 36)  print(rect.bottom\_y(), rect.top\_y())  print(rect.width(), rect.height())  print(rect.left\_x(), rect.right\_x())  print() | -15 13  -14 10  28 24  34 50  -21 13  -14 36  -14 78  68 92  -55 13 |

# Морской бой

Kлассная работа

макс. 3 балл.

|  |  |
| --- | --- |
| Ограничение времени | 1 секунда |
| Ограничение памяти | 64Mb |
| Ввод | стандартный ввод или input.txt |
| Вывод | стандартный вывод или output.txt |

Дима любит играть в морской бой. К сожалению, он очень рассеян и постоянно неправильно отмечает на карте клетки, по которым уже стрелял. Напишите класс, который будет строить за Диму карту.

Класс **SeaMap** должен иметь следующие методы (sm – экземпляр **SeaMap**):

sm.shoot(row, col, result) — добавить на карту результат выстрела, где  
row — индекс ряда карты,  
col — индекс вертикальной колонки карты,  
result — одна из строк: “miss” (промах), “hit” (попадание), “sink” (потопление корабля).

sm.cell(row, col), который  
возвращает ‘.’, если в клетке с координатами row, col может находиться корабль,  
возвращает ‘\*’, если в клетку уже стреляли или она находится рядом с потопленным кораблём,  
возвращает ‘x’ если в клетке было попадание.

Учтите, что не нужно помечать ‘\*’ клетки рядом с кораблём, в который попали, но не потопили до конца.

## Формат ввода

Каждый тест представляет собой код, в котором будет использоваться ваш класс. Файл c решением не обязательно называть solution.py, он будет переименован автоматически. Тест запускается с вашим классом, а его вывод сравнивается с правильным решением.

### Пример 1

| Ввод | Вывод |
| --- | --- |
| from solution import SeaMap  sm = SeaMap()  sm.shoot(2, 0, 'miss')  sm.shoot(6, 9, 'miss')  for row in range(10):  for col in range(10):  print(sm.cell(row, col), end='')  print() | ..........  ..........  \*.........  ..........  ..........  ..........  .........\*  ..........  ..........  .......... |

### Пример 2

| Ввод | Вывод |
| --- | --- |
| from solution import SeaMap  sm = SeaMap()  sm.shoot(2, 0, 'sink')  sm.shoot(6, 9, 'hit')  for row in range(10):  for col in range(10):  print(sm.cell(row, col), end='')  print() | ..........  \*\*........  x\*........  \*\*........  ..........  ..........  .........x  ..........  ..........  .......... |

### Пример 3

| Ввод | Вывод |
| --- | --- |
| from solution import SeaMap  sm = SeaMap()  sm.shoot(0, 0, 'sink')  sm.shoot(0, 9, 'sink')  sm.shoot(9, 0, 'sink')  sm.shoot(9, 9, 'sink')  for row in range(10):  for col in range(10):  print(sm.cell(row, col), end='')  print() | x\*......\*x  \*\*......\*\*  ..........  ..........  ..........  ..........  ..........  ..........  \*\*......\*\*  x\*......\*x |

## Примечания

Игровое поле имеет размер 10 на 10 клеток.

# Класс крестики-нолики

|  |  |
| --- | --- |
| Ограничение времени | 1 секунда |
| Ограничение памяти | 64Mb |
| Ввод | стандартный ввод или testmodule.py |
| Вывод | стандартный вывод или output.txt |

Напишите класс **TicTacToeBoard** для игры в крестики-нолики, который должен иметь следующие методы:

* **new\_game()** – для создания новой игры;
* **get\_field()** – для получения поля (список списков);
* **check\_field()** – для проверки, есть ли победитель, который возвращает **X**, если победил первый игрок, **0**, если второй, **D**, если ничья и **None**, если можно продолжать игру;
* **make\_move(row, col)** – который устанавливает значение текущего хода в ячейку поля с координатами **row, col**, если это возможно, «переключает» ход игрока, а также возвращает сообщение «Победил игрок X» при победе крестиков, «Победил игрок 0» при победе ноликов, «Ничья» в случае ничьей и «Продолжаем играть», если победитель после данного хода неопределён.

Кроме того, метод **make\_move** должен возвращать сообщение «Клетка <row>, <col> уже занята», если в клетке уже стоит крестик или нолик, и «Игра уже завершена», если в текущей игре уже выявлен победитель или закончились ячейки для ходов.

При создании объекта класса должна создаваться новая игра.  
Аргументы row и col метода make\_move могут принимать значения от 1 до 3.

## Пример

| Ввод | Вывод |
| --- | --- |
| from solution import TicTacToeBoard  board = TicTacToeBoard()  print(\*board.get\_field(), sep="\n")  print(board.make\_move(1, 1))  print(\*board.get\_field(), sep="\n")  print(board.make\_move(1, 1))  print(board.make\_move(1, 2))  print(\*board.get\_field(), sep="\n")  print(board.make\_move(2, 1))  print(board.make\_move(2, 2))  print(board.make\_move(3, 1))  print(board.make\_move(2, 2))  print(\*board.get\_field(), sep="\n") | ['-', '-', '-']  ['-', '-', '-']  ['-', '-', '-']  Продолжаем играть  ['X', '-', '-']  ['-', '-', '-']  ['-', '-', '-']  Клетка 1, 1 уже занята  Продолжаем играть  ['X', '0', '-']  ['-', '-', '-']  ['-', '-', '-']  Продолжаем играть  Продолжаем играть  Победил игрок X  Игра уже завершена  ['X', '0', '-']  ['X', '0', '-']  ['X', '-', '-'] |

## Примечания

Данная задача дополнительно проверяется преподавателем.