Общие положения

Общие положения Практикума по программированию.

За семестр будет 4 задания, каждое оценивается в 10 баллов. За просрочку сдачи (сдача после занятия-дедлайна) штраф -2 балла за каждое просроченное занятие, но максимальный штраф за просрочку -6 баллов (т.е. за задание студент получит 4 балла).

Списывание карается штрафом. Тотальное списывание – 0 баллов за задачу без права пересдачи. Локальная неспособность объяснить происходящее тоже штрафуется. Неполный функционал задачи при отсутствии желания доделывать – не полный балл за задачу. Тут главное каждому преподавателю установить единые критерии для своих групп. Позже возможно выработаем общие критерии.

Шкала:

35 баллов и выше – автомат без участия в зачете;

30-34 балла – зачет в виде собеседования (неформальный автомат);

Остальные приходят на зачет с целью получить итоговую сумму не менее 51 балла. На зачете будут решать задачи, которые выдавались в семестре, но с нуля и чужие варианты, при этом бальная оценка задач существенно возрастет (ориентировочно полная задача – 30 баллов). Т.е. для большинства это будет очень сложно. Цель: основной путь получения зачета своевременная сдача задач в течение семестра. Предполагается, что практикум станет точкой раннего отсева студентов неспособных/не желающих программировать. Я заранее уведомлю об этом деканат. Так что прошу точно вести записи о достижениях студентов, чтобы студенты к нам потом не придирались.

Задание 1

Т.к. решение первой задачи требует умения работать с циклами и списками прошу назначать дедлайн по первому заданию не ранее 3го занятия компьютерного практикума.

Для многих решение задачи будет выглядеть сложным. В связи с этим рекомендую показать, что решение задачи разбивается на несколько подзадач. Например, по первому заданию: получение и обработка ввода пользователя; отрисовка игрового поля; проверка условий и т.п. Можно предложить еще до сдачи полной задачи защитить отдельные подзадачи.

Задачи со звездочкой выдаются студентам только по желанию и одно и тоже задание со звездочкой выдается не более чем 2м студентам в группе. Я буду участвовать в приемке задания со звёздочкой (об этом нужно сразу им сообщить). Не нужно сильно запугивать, моя цель – не завалить их, а познакомиться с активом нашего потока.

Задачи для первого задания Практикума по программированию.

1. Реализовать программу, с которой можно играть в логическую игру **«Быки и коровы»** (описание правил игры: http://робомозг.рф/Articles/BullsAndCowsRules). Программа загадывает число, пользователь вводит очередной вариант отгадываемого числа, программа возвращает количество быков и коров и в случае выигрыша игрока сообщает о победе и завершается. Сама программа НЕ ходит, т.е. не пытается отгадать число загаданное игроком.
   Взаимодействие с программой производится через консоль, при запросе данных от пользователя программа сообщает, что ожидает от пользователя и проверяет корректность ввода.

2. Реализовать программу, при помощи которой 2 игрока могут играть в **«Крестики-нолики»** на поле 3 на 3. Взаимодействие с программой производится через консоль. Игровое поле изображается в виде трех текстовых строк и перерисовывается при каждом изменении состояния поля. При запросе данных от пользователя программа сообщает, что ожидает от пользователя (в частности, координаты новой отметки на поле) и проверяет корректность ввода. Программа должна уметь автоматически определять, что партия окончена, и сообщать о победе одного из игроков или о ничьей. Сама программа НЕ ходит, т.е. не пытается ставить крестики и нолики с целью заполнить линию.

3. Реализовать программу, при помощи которой 2 игрока могут играть в игру **«Супер ним»**. Правила игры следующие. На шахматной доске в некоторых клетках случайно разбросаны фишки или пуговицы. Игроки ходят по очереди. За один ход можно снять все фишки с какой-либо горизонтали или вертикали, на которой они есть. Выигрывает тот, кто заберет последние фишки. (описание правил игры: https://www.iqfun.ru/articles/super-nim.shtml)
   Взаимодействие с программой производится через консоль. Игровое поле изображается в виде текстовых строк и перерисовывается при каждом изменении состояния поля. При запросе данных от пользователя программа сообщает, что ожидает от пользователя (в частности, координаты новой отметки на поле) и проверяет корректность ввода. Программа должна уметь автоматически определять, что партия окончена, и сообщать о победе одного из игроков. Сама программа НЕ ходит, т.е. не пытается выбирать строки или столбцы с целью победить в игре.

4. Реализовать программу, с которой можно играть в игру **«19»**. Правила игры следующие. Нужно выписать подряд числа от 1 до 19: в строчку до 9, а потом начать следующую строку, в каждой клетке по 1 цифре (не числу (см пример по ссылке)). Затем игроку необходимо вычеркнуть парные цифры или дающие в сумме 10. Условие - пары должны находиться рядом или через зачеркнутые цифры по горизонтали или по вертикали. После того как все возможные пары вычеркнуты, оставшиеся цифры переписываются в конец таблицы. Цель - полностью вычеркнуть все цифры. (описание правил игры: http://podelki-fox.ru/igry-dlya-detey-na-bumage-s-chislami/)
   Взаимодействие с программой производится через консоль. Игровое поле изображается в виде трех текстовых строк и перерисовывается при каждом изменении состояния поля. При запросе данных от пользователя программа сообщает, что ожидает от пользователя (в частности, координаты очередного хода) и проверяет корректность ввода. Программа должна уметь автоматически определять, что нужно выписать новые строки с цифрами и то, что партия окончена. Сама программа НЕ ходит, т.е. не пытается выбирать пары цифр с целью окончить игру.

5. Реализовать программу, при помощи которой 3 игрока могут играть в игру **«Лоскутное одеяло»**. Правила игры следующие. На поле, имеющем размер 4 на 5 клеток за один ход каждый игрок должен заполнить одну клетку своим символом. Игрок старается, чтобы его символы были как можно дальше друг от друга. В ходе игры ведется подсчет очков: за каждое соседство клеток с одинаковыми символами игроку, владельцу символа добавляется одно штрафное очко. Соседними считаются клетки, имеющие общую сторону или расположенные наискосок друг от друга. Выигрывает тот, у кого в конце игры меньше всего штрафных очков.
   Взаимодействие с программой производится через консоль. Игровое поле изображается в виде 4 текстовых строк и перерисовывается при каждом изменении состояния поля. При запросе данных от пользователя программа сообщает, что ожидает от пользователя (например, координаты очередного хода) и проверяет корректность ввода. Программа должна уметь автоматически определять количество штрафных очков и окончание партии и ее победителя.
   Сама программа НЕ ходит, т.е. не пытается заполнять клетки символами с целью выиграть игру.

6. Реализовать программу, при помощи которой 2 игрока могут играть в игру **«Клондайк»**. Правила игры следующие. Игра ведётся на игровом поле размером 10 на 10 клеток. Игроки по очереди выставляют в любую свободную клетку по отметке, и тот игрок, после чьего хода получилась цепочка длиной хотя бы в 3 отметке, проигрывает. При этом в цепочке считаются как свои отметки, так и отметки соперника, у игровых фишек как бы нет хозяина. Цепочка - это ряд фишек, следующая фишка в котором примыкает к предыдущей с любого из 8-ми направлений. (описание правил игры: https://www.iqfun.ru/printable-puzzles/klondike-igra.shtml)
   Взаимодействие с программой производится через консоль. Игровое поле изображается в виде 10 текстовых строк и перерисовывается при каждом изменении состояния поля. При запросе данных от пользователя программа сообщает, что ожидает от пользователя (например, координаты очередного хода) и проверяет корректность ввода. Программа должна уметь автоматически определять окончание партии и ее победителя.
   Сама программа НЕ ходит, т.е. не пытается ставить в клетки отметки с целью выиграть игру.

7. Реализовать программу , при помощи которой 2 игрока могут играть в игру **«Максит»**. Правила игры следующие. В клетках квадрата 3 на 3 пишутся случайные числа из диапазона от 1 до 9. Начинающий выбирает любое понравившееся ему число и вычеркивает его, прибавляя к своей сумме. Второй игрок может выбрать любое из оставшихся чисел того столбца, в котором первый игрок делал свой предыдущий ход. Он тоже вычеркивает выбранное число, прибавляя его к своей сумме. Первый игрок далее поступает аналогично, выбирая число-кандидата из той строки, в которой второй игрок ходил перед этим. Может так случиться, что у какого-то игрока не будет хода. Тогда его соперник продолжает игру, делая ход в той же строке (для первого игрока) или в том же столбце (для второго игрока), что и до этого. Игра заканчивается, когда оба играющих не имеют ходов. Результат определяется по набранным суммам, у кого она больше, тот и выиграл. При равенстве сумм фиксируется ничья. (описание правил игры: https://www.iqfun.ru/articles/maxit.shtml).

Взаимодействие с программой производится через консоль. Игровое поле изображается в виде 3 текстовых строк и перерисовывается при каждом изменении состояния поля. При запросе данных от пользователя программа сообщает, что ожидает от пользователя (например, координаты очередного хода) и проверяет корректность ввода. Программа должна уметь автоматически определять сумму очков каждого из игроков и окончание партии и ее победителя.
Сама программа НЕ ходит, т.е. не пытается вычеркивать числа с целью выиграть игру.

8. (*) Реализовать программу, при помощи которой 2 игрока могут играть в игру **«Мостики»**. Правила игры следующие. В ходе игры каждый из игроков старается построить мост с одного своего берега на другой по камням, образующим массив 4 на 5 (4 камня вдоль берега игрока и 5 камней между берегами). У первого игрока - крестики в качестве камней и берега крестиков (левый и правый край поля), у второго игрока – нолики и берега ноликов (верхний и нижний край поля). Игру можно начинать в любой точке поля. За один ход игрок может соединить два своих соседних камня вертикальным или горизонтальным мостиком (обозначаются в текстовом режиме символами «-» и «|»). Мосты первого и второго игрока пересекаться не должны. Выигрывает тот, кто построит непрерывный мост с одного своего берега на другой. (описание правил игры: https://www.7ya.ru/article/Chem-zanyat-rebenka-13-igr-na-liste-bumagi-so-slovami-kartinkami/)
Взаимодействие с программой производится через консоль. Игровое поле изображается в виде 9 текстовых строк и перерисовывается при каждом изменении состояния поля. При запросе данных от пользователя программа сообщает, что ожидает от пользователя (например, координаты очередного хода) и проверяет корректность ввода. Программа должна уметь автоматически определять недопустимые ходы (приводящие к пересечению мостов соперников) и окончание партии и ее победителя.
Сама программа НЕ ходит, т.е. не пытается строить мосты с целью выиграть игру.

9. (*) Реализовать программу, с которой можно играть в игру **«Морской бой»**. Программа автоматически случайно расставляет на поле размером 10 на 10 клеток: 4 1-палубных корабля, 3 2-палубных корабля, 2 3-палубных корабля и 1 4-х палубный. Между любыми двумя кораблями по горизонтали и вертикали должна быть как минимум 1 незанятая клетка. Программа позволяет игроку ходить, производя выстрелы. Сама программа НЕ ходит т.е. не пытается топить корабли расставленные игроком.
Взаимодействие с программой производится через консоль. Игровое поле изображается в виде 10 текстовых строк и перерисовывается при каждом изменении состояния поля. При запросе данных от пользователя программа сообщает, что ожидает от пользователя (в частности, координаты очередного «выстрела») и проверяет корректность ввода. Программа должна уметь автоматически определять потопление корабля и окончание партии и сообщать об этих событиях.

10. (*) Реализовать программу, с которой можно играть в игру **«Пятнашки»**. Правила игры следующие.
Головоломка представляет собой 15 квадратных костяшек с числами от 1 до 15. Все костяшки заключены в квадратную коробку (поле) размером 4 на 4. При размещении костяшек в коробке остается одно пустое место, которое можно использовать для перемещения костяшек внутри коробки. Цель игры - упорядочить размещение чисел в коробке, разместив их по возрастанию слева направо и сверху вниз, начиная с костяшки с

номером 1 в левом верхнем углу и заканчивая пустым местом в правом нижнем углу коробки.
Взаимодействие с программой производится через консоль. Игровое поле изображается в виде 4 текстовых строк и перерисовывается при каждом изменении состояния поля. При запросе данных от пользователя программа сообщает, что ожидает от пользователя (например, координаты очередного хода) и проверяет корректность ввода. Программа должна считать количество сделанных ходов, уметь автоматически определять недопустимые ходы, окончание партии и ее победителя.
Сама программа НЕ ходит, т.е. не пытается упорядочить костяшки с целью выиграть игру.

Задание 2

Задачи для второго задания Практикума по программированию. Общая тема задания «текстовый калькулятор».

Базовая часть (выполняется всеми самостоятельно!):

Написать калькулятор для строковых выражений вида '<число> <операция> <число>', где <число> - не отрицательное целое число меньшее 100, записанное словами, например "тридцать четыре", <арифмитическая операция> - одна из операций "плюс", "минус", "умножить". Результат выполнения операции вернуть в виде текстового представления числа. Пример calc("двадцать пять плюс тринадцать") -> "тридцать восемь"

Оформить калькулятор в виде функции, которая принимает на вход строку и возвращает строку.

Дополнительные задания:

1) Реализовать поддержку операции деления и остатка от деления и работу с дробными числами (десятичными дробями). Пример: calc("сорок один и тридцать одна сотая разделить на семнадцать") -> "два и сорок три сотых". Обрабатывать дробную часть до тысячных включительно, если при делении получаются числа с меньшей дробной частью выполнять округление до тысячных.
   Сложность 2
2) *Расширение задания 1.* Реализовать поддержку десятичной дробной части до миллионных долей включительно. Реализовать корректный вывод информации о периодической десятичной дроби (период дроби вплоть до 4х десятичных знаков). Пример: calc("девятнадцать и восемьдесят две сотых разделить на девяносто девять") -> "ноль и двадцать сотых и ноль два в периоде ".
   Сложность 3
3) Реализовать текстовый калькулятор для выражения из произвольного количества операций с учетом приоритета операций. Пример: calc("пять плюс два умножить на три минус один") -> "десять". (Для реализации рекомендуется использовать алгоритмы основанные на польской инверсной записи см. например, https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%80%D0%B0%D1%82%D0%BD%D0%B

0%D1%8F_%D0%BF%D0%BE%D0%BB%D1%8C%D1%81%D0%BA%D0%B0%D1%8F_%D0%B7%D0%B0%D0%BF%D0%B8%D1%81%D1%8C)

Сложность 3

4) *Расширение задания 3.* Добавить поддержку приоритета операций с помощью скобок. Пример: calc("скобка открывается пять плюс два скобка закрывается умножить на три минус один") -> "двадцать".
   Сложность 3
5) Добавить возможность использования отрицательных чисел. Пример: calc("пять минус минус один") -> "шесть".
   Сложность 1
6) Добавить возможность оперировать с дробями (правильными и смешанными). Реализовать поддержку сложения, вычитания и умножения, дробей. Результат операций не должен представлять неправильную дробь, такие результаты нужно превращать в смешанные дроби. Пример: calc("один и четыре пятых плюс шесть седьмых ") -> "два и двадцать три тридцать пятых".
   Сложность 3
7) *Расширение задания 6.* Добавить автоматическое сокращение дроби в ответе. Пример: calc("одна шестая умножить на две третьих") -> "одна девятая".
   Сложность 1
8) *Расширение задания 1.* Добавить операции возведения в степень и тригонометрические операции синус, косинус, тангенс и константу пи. Допускается как минимум одна из этих функций в выражении с обычными операциями. Пример: calc("два в степени четыре") -> "шестнадцать". Пример: calc("синус от пи разделить на четыре") -> "ноли и семьсот семь тысячных".
   Сложность 1 или 2
9) Добавить комбинаторные операции перестановки, размещения и сочетания. Пример: calc("размещений из трех по два") -> "шесть".
   Сложность 1 или 2
10) Диагностировать ошибки: неправильную запись числа; неправильную последовательность чисел и операций; (задание 1) деление на ноль; (задание 3) неправильную последовательность чисел и операций; (задание 4) некорректный баланс и вложенность скобок; (задание 6) некорректную запись числа
    Сложность 1 или 2

Для каждого студента формируется комплексное задание из сочетания пунктов. Суммарная сложность комплексного задания должна быть не менее 7. Все выбранные пункты должны работать в функции calc() одновременно.

Приветствуется выполнение заданий с суммарной сложностью более 7 (рекомендуется отмечать такие решения дополнительными баллами). Если реализовано 2 и более пунктов сложностью 3, то, считается, что выполнено задание со звездочкой.

Сложность пункта с интервальной сложностью (например: Сложность 1 или 2) определяется преподавателем в зависимости от сочетания этого пункта с другими пунктами, выбранными студентом. В случае неоднозначности сочетания требований двух пунктов студент предлагает преподавателю свое видение итоговой постановки задачи, а преподаватель согласует его (рекомендуется фиксировать договоренность в письменном виде). Преподаватель имеет право увеличить оценку сложности

комплексного задания в случае существенного увеличения трудоемкости при сочетании двух пунктов.

Преподаватель имеет право добавить пункты с собственной постановкой задачи и оценкой сложности. Студент может предложить свои пункты и реализовывать их в случае предварительно согласования с преподавателем постановки задачи и оценки сложности пункта. Рекомендуется фиксировать дополнительные пункты письменно. Желательно, чтобы дополнительные пункты составляли не более половины суммарной трудоемкости комплексного задания.

Задание 3

Задачи для третьего задания Практикума по программированию. Общая тема задания «реализация собственного пакета модулей по манипулированию табличными данными».

Базовая часть (выполняется всеми самостоятельно!):

На базе модулей: csv, pickle и прямой работы с файлами реализовать следующий базовый функционал:

1. функций **load_table**, **save_table** по загрузке/сохранению табличных данных во внутреннее представление модуля/из внутреннего представления модуля:
    - файла формата csv (отдельный модуль с **load_table**, **save_table** в рамках общего пакета)
    - файла формата pickle (отдельный модуль с **load_table**, **save_table** в рамках общего пакета), модуль использует структуру данных для представления таблицу, удобную автору работы.
    - текстового файла (только функция **save_table** сохраняющая в текстовом файле представление таблицы, аналогичное выводу на печать с помощью функции **print_table()).**

Примечание: внутреннее представление может базироваться на словаре, где по разным ключам хранятся ключевые «атрибуты» таблицы, а значения таблицы хранятся в виде вложенных списков. Студент может выбрать другое внутреннее представление таблицы (согласовав его с преподавателем), в том числе, студенты знакомые с ООП на Python, могут реализовать собственный класс для таблицы.

При определении api модулей максимально полно использовать возможности сигнатур функций на Python (значения по умолчанию, запаковка/распаковка, в т.ч. именованных параметров, возвращение множественных значений), интенсивно выполнять проверки и возбуждать исключительные ситуации.

2. модуля с базовыми операциями над таблицами:
    - **get_rows_by_number(start, [stop], copy_table=False)** – получение таблицы из одной строки или из строк из интервала по номеру строки. Функция либо копирует исходные данные, либо создает новое представление таблицы, работающее с

исходным набором данных (**copy_table=False**), таким образом изменения, внесенные через это представления будут наблюдаться и в исходной таблице.

- **get_rows_by_index(val1, … , copy_table=False)** – получение новой таблицы из одной строки или из строк со значениями в первом столбце, совпадающими с переданными аргументами **val1, … , valN**. Функция либо копирует исходные данные, либо создает новое представление таблицы, работающее с исходным набором данных (**copy_table=False**), таким образом изменения, внесенные через это представления будут наблюдаться и в исходной таблице.
- **get_column_types(by_number=True)** – получение словаря вида *столбец:тип_значений*. Тип значения: int, float, bool, str (по умолчанию для всех столбцов). Параметр **by_number** определяет вид значения столбец – целочисленный индекс столбца или его строковое представление.
- **set_column_types(types_dict, by_number=True)** – задание словаря вида *столбец:тип_значений*. Тип значения: int, float, bool, str (по умолчанию для всех столбцов). Параметр **by_number** определяет вид значения столбец – целочисленный индекс столбца или его строковое представление.
- **get_values(column=0)** – получение списка значений (типизированных согласно типу столбца) таблицы из столбца либо по номеру столбца (целое число, значение по умолчанию 0, либо по имени столбца)
- **get_value(column=0) –** аналог **get_values(column=0)** для представления таблицы с одной строкой, возвращает не список, а одно значение (типизированное согласно типу столбца).
- **set_values(values, column=0)** – задание списка значений **values** для столбца таблицы (типизированных согласно типу столбца) либо по номеру столбца (целое число, значение по умолчанию 0, либо по имени столбца).
- **set_value(column=0) –** аналог **set_values(value, column=0)** для представления таблицы с одной строкой, устанавливает не список значений, а одно значение (типизированное согласно типу столбца).
- **print_table()** – вывод таблицы на печать.

3. Для каждой функции должно быть реализована генерация не менее одного вида исключительных ситуаций.

Дополнительные задания:

1) В **load_table** реализовать **load_table(file1, …)** – поддержку загрузки таблицы, разбитой на несколько файлов (произвольное количество фйалов) (для форматов csv и pickle). В случае несоответствия структуры столбцов файлов вызывать исключительную ситуацию.
Сложность 1

2) *Расширение задания 1.*
В **save_table** реализовать поддержку сохранения таблицы в разбитой на несколько файлов (произвольное количество фйалов) по параметру **max_rows**, определяющему максимальное количество строк в файле. Файлы csv и pickle, полученные с помощью **save_table** должны быть совместимы с load_table из задания 1.
Сложность 1

3) Реализовать функцию **concat(table1, table2)** и **split(row_number)** склеивающую две таблицы или разбивающую одну таблицу на 2 по номеру строки.
Сложность 1

4) Реализовать автоматическое определение типа столбцов по хранящимся в таблице значениям. Оформить как отдельную функцию и встроить этот функционал как опцию работы функции **load_table**.
Сложность 1 или 2

5) Реализовать поддержку дополнительного типа значений «дата и время» на основе модуля datetime.
Сложность 1 или 2

6) Добавить набор функций **add**, **sub**, **mul**, **div**, которые обеспечат выполнение арифмитических операций для столбцов типа int, float, bool. Продумать сигнатуру функций и изменения в другие функции, которые позволят удобно выполнять арифметические операции со столбцами и присваивать результаты выч. Реализовать реагирование на некорректные значения с помощью генерации исключительных ситуаций.
Сложность 2

7) По аналогии с п. 6 реализовать функции **eq (==), gr (>), ls (<), ge (>=), le (<=), ne (==), которые** возвращают список булевских значений длинной в количество строк сравниваемых столбцов. Реализовать функцию **filter_rows (bool_list, copy_table=False)** – получение новой таблицы из строк для которых в **bool_list** (длинной в количество строк в таблице) находится значение True.
Сложность 3

8) Реализовать функцию **merge_tables(table1, table2, by_number=True)**: в результате слияния создается таблица с набором столбцов, соответствующих объединенному набору столбцов исходных таблиц. Соответствие строк ищется либо по их номеру (**by_number=True**) либо по значению индекса (1й столбец). При выполнении слияния возможно множество конфликтных ситуаций. Автор должен их описать и определить допустимый способ реакции на них (в т.ч. через дополнительные параметры функции и инициацию исключительных ситуаций).
Сложность 2

9) Реализовать полноценную поддержку значения **None** в незаполненных ячейках таблицы. Должно работать при загрузке ячеек с пропусками значений, при операциях приводящих к появлению пустых ячеек, при работе с get и set операциями.
Сложность 1 или 2

Для каждого студента формируется комплексное задание из сочетания пунктов. Суммарная сложность комплексного задания должна быть не менее 6. Все выбранные пункты должны быть в виде модулей, лежащих в одном пакете. Для каждого вида функционала (в том числе проверок и следующих за ними исключительных ситуаций) должен быть реализован пример в jupyther notebook.

Приветствуется выполнение заданий с суммарной сложностью более 6 (рекомендуется отмечать такие решения дополнительными баллами).

Сложность пункта с интервальной сложностью (например: Сложность 1 или 2) определяется преподавателем в зависимости от сочетания этого пункта с другими пунктами, выбранными студентом. В случае неоднозначности сочетания требований двух пунктов студент предлагает преподавателю свое видение итоговой постановки задачи, а преподаватель согласует его (рекомендуется фиксировать договоренность в письменном виде). Преподаватель имеет право увеличить оценку сложности комплексного задания в случае существенного увеличения трудоемкости при сочетании двух пунктов.

Преподаватель имеет право добавить пункты с собственной постановкой задачи и оценкой сложности. Студент может предложить свои пункты и реализовывать их в случае предварительно согласования с преподавателем постановки задачи и оценки сложности пункта. Рекомендуется фиксировать дополнительные пункты письменно. Желательно, чтобы дополнительные пункты составляли не более половины суммарной трудоемкости комплексного задания

Задание 4

Задачи для четвертого задания Практикума по программированию. Общая тема задания «Шахматный симулятор».

Базовая часть (выполняется всеми самостоятельно!):

Реализовать программу, которая позволяет играть в шахматы на компьютере. Взаимодействие с программой производится через консоль (базовый вариант). Игровое поле изображается в виде 8 текстовых строк, плюс строки с буквенным обозначением столбцов (см. пример на Рис. 1) и перерисовывается при каждом изменении состояния поля. При запросе данных от пользователя программа сообщает, что ожидает от пользователя (например, позицию фигуры для следующего хода белыми; целевую позицию выбранной фигуры) и проверяет корректность ввода (допускаются только ходы соответствующие правилам шахмат; поддержка рокировки, сложных правил для пешек и проверки мата вынесена в отдельные пункты). Программа должна считать количество сделанных ходов.

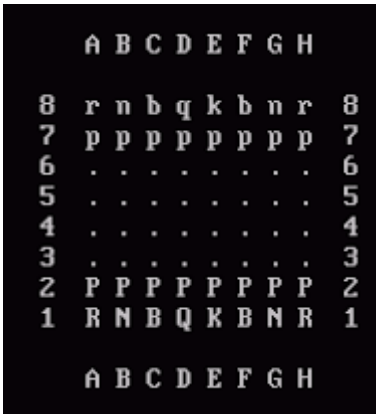


Рис. 1 Пример изображения шахматного поля в текстовом режиме

Сама программа НЕ ходит: т.е. не пытается выполнить ходы за одну из сторон, а предоставляет поочередно вводить ходы за белых и черных.

Дополнительные задания:

Справка о шахматной нотации:

1. Общая информация о шахматной нотации записи партий: https://ru.wikipedia.org/wiki/Шахматная_нотация
2. Партии в полной нотации: бесплатная база (для открытия партий нужно зарегистрироваться на ресурсе) записей партий в шахматной нотации (полной): http://www.chessebook.com/openings.php?lan=ru&pa=pa (для получения файлов с записью партий копируйте текст понравившихся партий в текстовый файл, скопированный текст не подвергать дополнительному редактированию и сохранить в файл).
3. Партии в сокращенной нотации берем из обсуждений на kasparovchess.crestbook.com, например, из этой ветки: http://kasparovchess.crestbook.com/threads/8210/ (для получения файлов с записью партий копируйте текст понравившихся партий, расположенных

справа от блока с доской, в текстовый файл, скопированный текст не подвергать дополнительному редактированию (он во многих нюансах будет отличаться от партий с chessebook.com, так и должно быть) и сохранять файл).

1. Реализовать чтение записи шахматной партии из выбранного пользователем файла в полной нотации. После чтения должна быть возможность двигаться вперед и назад по записи партии (с соответствующим изменением на поле). Должна быть возможность в выбранной позиции перейти из режима просмотра партии в обычный режим игры. Протестировать не менее чем на 20 реальных партиях с сайта.
   Сложность 2
2. Реализовать чтение записи шахматной партии из выбранного пользователем файла в сокращенной нотации. После чтения должна быть возможность двигаться вперед и назад по записи партии (с соответствующим изменением на поле). Должна быть возможность в выбранной позиции перейти из режима просмотра партии в обычный режим игры. Протестировать не менее чем на 20 реальных партиях с сайта.
   Сложность 3 (если пункты 1 и 2 совместно, то суммарная сложность 4)
3. Реализовать возможность записи разыгрываемой шахматной партии в текстовый файл в полной (сокращенной, если студент выполнял задание 2) нотации. Записать партию можно на любом ходу, с историей всей партии с самого начала. Записанная партия должна корректно воспроизводиться в режиме чтения записи партии.
   Сложность 2
4. Реализовать возможность «отката» ходов. С помощью специальной команды можно возвращаться на ход (или заданное количество ходов) назад вплоть до начала партии.
   Сложность 1
5. Реализовать функцию подсказки выбора новой позиции фигуры: после выбора фигуры для хода функция визуально на поле показывает поля доступные для хода или фигуры соперника, доступные для взятия, выбранной фигурой.
   Сложность 1
6. Реализовать функцию подсказки угрожаемых фигур: она возвращает информацию о том, какие фигуры ходящего игрока сейчас находятся под боем (т.е. могут быть взяты соперником на следующий ход) и визуально выделяет их на поле. Функция отдельно указывает на наличие шаха королю.
   Сложность 1
7. Автоматически определять мат (правила определения: https://ru.wikipedia.org/wiki/Мат_(шахматы)).
   Сложность 2
8. Реализовать поддержку выполнения рокировки по всем шахматным правилам (в базовой версии поддержка рокировки не обязательна). Правила рокировки см.: https://ru.wikipedia.org/wiki/Рокировка
   Сложность 1
9. Реализовать поддержку для пешки сложных правил: «взятие на проходе» и замены на другух фигуру при достижении крайней горизонтали (в базовой версии их поддержка не обязательна, но возможность первого хода на одну или две горизонтали - обязательно). Подробнее о правилах см.: https://ru.wikipedia.org/wiki/Правила_шахмат .
   Сложность 1

Для каждого студента формируется комплексное задание из сочетания пунктов. Суммарная сложность комплексного задания должна быть не менее 5, как минимум

одна задача из комплекта должна стоить дороже 1. Приветствуется выполнение заданий с суммарной сложностью более 5 (рекомендуется отмечать такие решения дополнительными баллами).

Сложность комплексного задания может быть скорректирована преподавателем в зависимости от сочетания пунктов между собой. В случае неоднозначности сочетания требований двух пунктов студент предлагает преподавателю свое видение итоговой постановки задачи, а преподаватель согласует его (рекомендуется фиксировать договоренность в письменном виде). Преподаватель имеет право увеличить оценку сложности комплексного задания в случае существенного увеличения трудоемкости при сочетании двух пунктов.

Преподаватель имеет право добавить пункты с собственной постановкой задачи и оценкой сложности. Студент может предложить свои пункты и реализовывать их в случае предварительно согласования с преподавателем постановки задачи и оценки сложности пункта. Рекомендуется фиксировать дополнительные пункты письменно. Желательно, чтобы дополнительные пункты составляли не более половины суммарной трудоемкости комплексного задания

Задание 5

Задачи для пятого задания Практикума по программированию. Общая тема задания «Шахматный симулятор: объектно-ориентированная версия».

Базовая часть (выполняется всеми самостоятельно!):

На базе собственной реализации Задания 4 создать объектно-ориентированную реализацию программы для игры в шахматы.

Базовые требования к функциональности программы сохраняются прежними:
Реализовать программу, которая позволяет играть в шахматы на компьютере. Взаимодействие с программой производится через консоль (базовый вариант). Игровое поле изображается в виде 8 текстовых строк, плюс строки с буквенным обозначением столбцов (см. пример на Рис. 1) и перерисовывается при каждом изменении состояния поля. При запросе данных от пользователя программа сообщает, что ожидает от пользователя (например, позицию фигуры для следующего хода белыми; целевую позицию выбранной фигуры) и проверяет корректность ввода (допускаются только ходы соответствующие правилам шахмат; поддержка рокировки, сложных правил для пешек и проверки мата вынесена в отдельные пункты). Программа должна считать количество сделанных ходов.

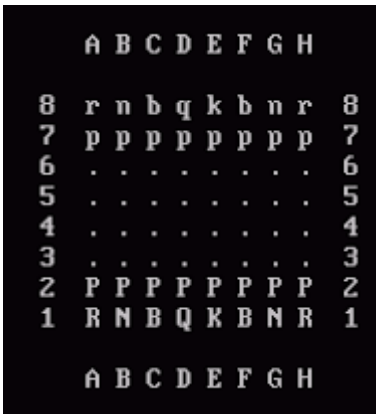


Рис. 1 Пример изображения шахматного поля в текстовом режиме

Сама программа НЕ ходит: т.е. не пытается выполнить ходы за одну из сторон, а предоставляет поочередно вводить ходы за белых и черных.

Требования к реализации:
Основные объекты и абстрактные сущности игры должны быть представлены в виде объектов, представителей соответствующих классов, часть классов должны быть организованы в виде иерархии. В частности: шахматные фигуры – объекты, представители классов, организованных в виде иерархии; доска – объект; ходы фигур – объекты. Вся основная информация должна храниться в атрибутах объектов или классов (например, информация о положении фигур, цвете фигур, символах, используемых для визуализации фигур и т.п.). Основная часть функционала должна программы должна быть организована в виде методов, закрепленных за соответствующими объектами или классами. Например, это касается методов определяющих

допустимые ходы фигур. Организация иерархий классов, атрибутов и методов должна позволять гибко расширять возможности программы с минимальными изменениям в уже созданном коде.

Дополнительные задания:

1. Придумать 3 новых вида фигур с оригинальными правилами перемещения и реализовать их классы. Создать модификацию шахмат с новыми фигурами с минимальным вмешательством в существующий код.
   Сложность 1
2. На базе игры в шахматы реализовать игру в шашки. Разработать модификацию шахмат с минимальным вмешательством в существующий код.
   Сложность 2
3. На базе игры в шахматы на классической доске реализовать игру в гексагональные шахматы (https://ru.wikipedia.org/wiki/Гексагональные_шахматы). Выбрать один из трех вариантов: шахматы Глинского; шахматы МакКуэя; шахматы Шафрана. Разработать модификацию шахмат с минимальным вмешательством в существующий код для обычных шахмат.
   Сложность 3
4. На базе игры в шахматы на классической доске реализовать игру в гексагональные шахматы на троих (https://ru.wikipedia.org/wiki/Шахматы_для_троих). Выбрать один из существующих вариантов. Разработать модификацию шахмат с минимальным вмешательством в существующий код для обычных шахмат.
   Сложность 4
5. Реализовать возможность «отката» ходов. С помощью специальной команды можно возвращаться на ход (или заданное количество ходов) назад вплоть до начала партии. Информация о ходах в партии должна храниться в объектно-ориентированном виде.
   Сложность 1
6. Реализовать функцию подсказки выбора новой позиции фигуры: после выбора фигуры для хода функция визуально на поле показывает поля доступные для хода или фигуры соперника, доступные для взятия, выбранной фигурой. Информация о допустимых ходах должна храниться в объектно-ориентированном виде, алгоритм без модификации должен работать при добавлении новых типов фигур (задание берется совместно с Заданием 1).
   Сложность 1
7. Реализовать функцию подсказки угрожаемых фигур: она возвращает информацию о том, какие фигуры ходящего игрока сейчас находятся под боем (т.е. могут быть взяты соперником на следующий ход) и визуально выделяет их на поле. Функция отдельно указывает на наличие шаха королю. Информация о допустимых ходах должна храниться в объектно-ориентированном виде, алгоритм без модификации должен работать при добавлении новых типов фигур (задание берется совместно с Заданием 1).
   Сложность 1
8. Реализовать поддержку для пешки сложных правил: «взятие на проходе» и замены на другух фигуру при достижении крайней горизонтали (в базовой версии их поддержка не обязательна, но возможность первого хода на одну или две горизонтали - обязательно). Подробнее о правилах см.: https://ru.wikipedia.org/wiki/Правила_шахмат . Информация о допустимых ходах должна храниться в объектно-ориентированном виде, алгоритм без модификации должен работать при добавлении новых типов фигур со сложным поведением (задание берется совместно с Заданием 1 и как минимум одна из новых фигур

должна иметь сложное поведение, т.е. изменение правил хода и взятия фигуры в зависимости от дополнительных условий).

Сложность 1

Для каждого студента формируется комплексное задание из сочетания пунктов. Суммарная сложность комплексного задания должна быть не менее 5, как минимум одна задача из комплекта должна стоить дороже 1. Приветствуется выполнение заданий с суммарной сложностью более 5 (рекомендуется отмечать такие решения дополнительными баллами).

Сложность комплексного задания может быть скорректирована преподавателем в зависимости от сочетания пунктов между собой. В случае неоднозначности сочетания требований двух пунктов студент предлагает преподавателю свое видение итоговой постановки задачи, а преподаватель согласует его (рекомендуется фиксировать договоренность в письменном виде). Преподаватель имеет право увеличить оценку сложности комплексного задания в случае существенного увеличения трудоемкости при сочетании двух пунктов.

Преподаватель имеет право добавить пункты с собственной постановкой задачи и оценкой сложности. Студент может предложить свои пункты и реализовывать их в случае предварительно согласования с преподавателем постановки задачи и оценки сложности пункта. Рекомендуется фиксировать дополнительные пункты письменно. Желательно, чтобы дополнительные пункты составляли не более половины суммарной трудоемкости комплексного задания

Задание 6

Задачи для шестого задания Практикума по программированию. Общая тема задания «реализация собственного пакета модулей для манипулирования плоскими фигурами».

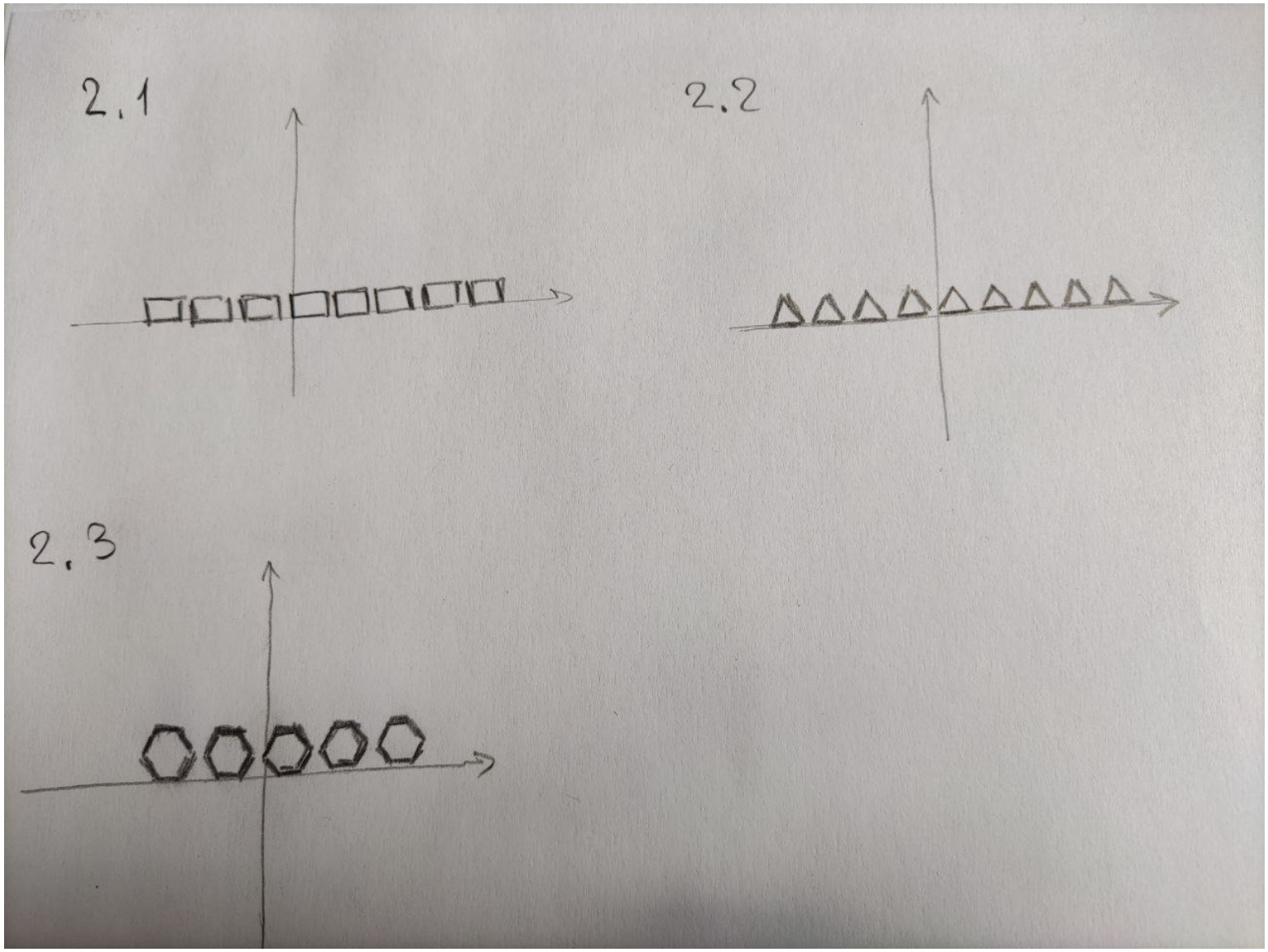
Базовые требования к функциональности программы сохраняются прежними:

Реализовать api, которое позволяет: генерировать, преобразовывать и визуализировать последовательность плоских полигонов, представленных в виде картежа картежей (например: ((0,0), (0,1), (1,1), (1,0)) – представление для квадрата). Последовательности представлений полигонов представляют из себя итераторы (далее: последовательности полигонов). Решать задачи с использованием функционального стиля программирования, в том числе активно использовать функции из модуля itertools и functools.

1. Реализовать функцию визуализации последовательности полигонов, представленной в виде итератора (например, можно использовать визуализацию с помощью библиотеки matplotlib, см. пример): https://matplotlib.org/stable/gallery/shapes_and_collections/patch_collection.html#sphx-glr-gallery-shapes-and-collections-patch-collection-py

   (обязательная часть)
2. Реализовать функции, генерирующие бесконечную последовательность не пересекающихся полигонов с различающимися координатами (например, «ленту»):
   1. прямоугольников (**gen_rectangle()**);
   2. треугольников (**gen_triangle()**);
   3. правильных шестиугольников (**gen_hexagon()**).
   4. с помощью данных функций используя функции из модуля itertools сгенерировать 7 фигур, включающих как прямоугольники, так и треугольники и шестиугольники, визуализировать результат.

   (обязательная часть)

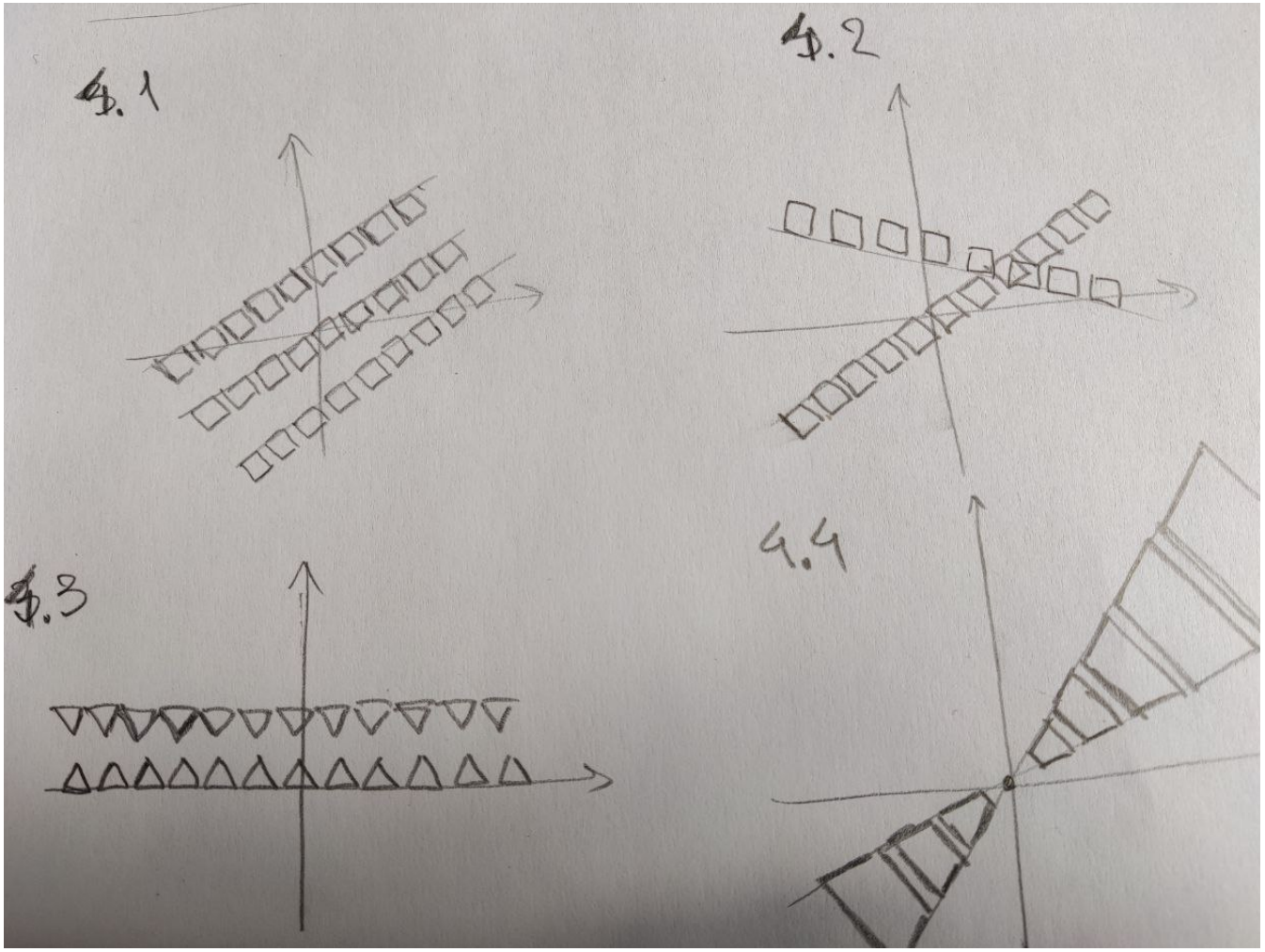


3. Реализовать операции:
    1. параллельный перенос (**tr_translate**);
    2. поворот (**tr_rotate**);
    3. симметрия (**tr_symmetry**);

4. гомотетия (**tr_homothety**);

которые можно применить к последовательности полигонов с помощью функции map.

(обязательная часть)



4. С помощью данных функций создать и визуализировать:
    1. 3 параллельных «ленты» из последовательностей полигонов, расположенных под острым углом к оси x;
    2. две пересекающихся «ленты» из последовательностей полигонов, пересекающихся не в начале координат;
    3. две параллельных ленты треугольников, ориентированных симметрично друг к другу;
    4. последовательность четырехугольников в разном масштабе, ограниченных двумя прямыми, пересекающимися в начале координат (см. рис.).

(обязательная часть)

5. Реализовать операции:
    1. фильтрации фигур, являющихся выпуклыми многоугольниками (**flt_convex_polygon**);
    2. фильтрации фигур, имеющих хотя бы один угол, совпадающий с заданной точкой (**flt_angle_point**);
    3. фильтрации фигур, имеющих площадь менее заданной (**flt_square**);
    4. фильтрации фигур, имеющих кратчайшую сторону менее заданного значения (**flt_short_side**);
    5. фильтрации выпуклых многоугольников, включающих заданную точку (внутри многоугольника) (**flt_point_inside**);
    6. фильтрации выпуклых многоугольников, включающих любой из углов заданного многоугольника (**flt_polygon_angles_inside**);

    которые можно применить к последовательности полигонов с помощью функции filter.

(обязательная часть: 2 пункта, 4 пункта – сложность 1; 6 пунктов – сложность 2)

6. С помощью данных функций реализовать и визуализировать:
    1. фильтрацию фигур, созданных в рамках пункта 4.4; подобрать параметры так, чтобы на выходе было получено 6 фигур;
    2. используя функции генерации из п. 2 и операции из п. 3 создать не менее 15 фигур, которые имеют различный масштаб и выбрать из них (подбором параметра фильтрации) не более 4х фигур, имеющих кратчайшую сторону менее заданного значения;
    3. используя функции генерации из п. 2 и операции из п. 3 создать не менее 15 фигур имеющих множество пересечений и обеспечить фильтрацию пересекающихся фигур.

(обязательная часть: 1 пункт, 3 пункта – сложность 1)

7. Реализовать декораторы и продемонстрировать корректность их работы:
    1. Фильтрующие многоугольники в итераторах среди аргументов функции, работающие на основе функций из 5: **@flt_convex_polygon, @flt_angle_point, @flt_square, @flt_short_side, @flt_point_inside, @flt_polygon_angles_inside** ;
    2. Преобразующие многоугольники в итераторах среди аргументов функции, работающие на основе функций из 3: **@tr_translate, @tr_rotate, @tr_symmetry, @tr_homothety** ;

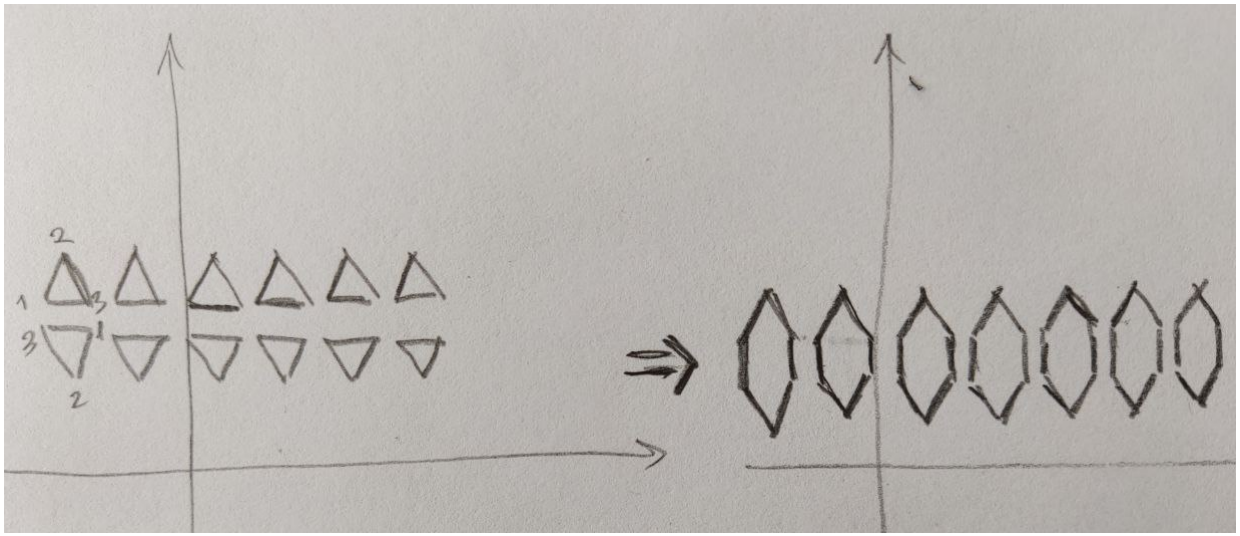
(обязательная часть: 1 пункта, 5 пунктов – сложность 1)

8. Реализовать функции и продемонстрировать их корректность:
    1. поиск угла, самого близкого к началу координат (**agr_origin_nearest**);
    2. поиск самого длинной стороны многоугольника (**agr_max_side**);
    3. поиск самой маленькой площади многоугольника (**agr_min_area**);
    4. расчет суммарного периметра (**agr_perimeter**);
    5. расчет суммарной площади (**agr_area**).

    которые можно применить к последовательности полигонов с помощью функции functools.reduce .

(3 пункта – сложность 1; 5 пунктов – сложность 2)

9. Реализовать функции и продемонстрировать пример их работы (если возможно, с визуализацией):
    1. склейки полигонов в одну последовательность полигонов из нескольких последовательностей полигонов **zip_polygons(iterator1, iterator2, [iterator3, …])**. Пример: **zip_polygons([((1,1), (2,2), (3,1)), ((11,11), (12,12), (13,11))], [((1,-1), (2,-2), (3,-1)), ((11,-11), (12,-12), (13,-11))])** ->[((1,1), (2,2), (3,1), (1,-1), (2,-2), (3,-1)), ((11,11), (12,12), (13,11), (11,-11), (12,-12), (13,-11))] .
    Альтернативный пример (визуализация):



2. генерации **count_2D()** параметры: (start1, start2), [(step1, step2)], результаты: (start1, start2), (start1+step1, start2+step2), (start1+2*step1, start2+2*step2)
3. склейки полигонов в одну последовательность полигонов из нескольких последовательностей **zip_tuple(iterator1, iterator2) . Пример: zip_tuple([(1,1), (2,2), (3,3), (4,4)], [(2,2), (3,3), (4,4), (5,5)], [(3,3), (4,4), (5,5), (6,6)]) ->** ((1,1), (2,2), (3,3)), ((2,2), (3,3) (4,4)), ((3,3), (4,4), (5,5)), ((5,5), (6,6), (7,7))

(3 пункта – сложность 1)

Для каждого студента формируется комплексное задание из сочетания пунктов. Суммарная сложность комплексного задания должна быть не менее 5, как минимум одна задача из комплекта должна стоить дороже 1. Приветствуется выполнение заданий с суммарной сложностью более 5 (рекомендуется отмечать такие решения дополнительными баллами).

Сложность комплексного задания может быть скорректирована преподавателем в зависимости от сочетания пунктов между собой. В случае неоднозначности сочетания требований двух пунктов студент предлагает преподавателю свое видение итоговой постановки задачи, а преподаватель согласует его (рекомендуется фиксировать договоренность в письменном виде). Преподаватель имеет право увеличить оценку сложности комплексного задания в случае существенного увеличения трудоемкости при сочетании двух пунктов.

Преподаватель имеет право добавить пункты с собственной постановкой задачи и оценкой сложности. Студент может предложить свои пункты и реализовывать их в случае предварительно согласования с преподавателем постановки задачи и

оценки сложности пункта. Рекомендуется фиксировать дополнительные пункты письменно. Желательно, чтобы дополнительные пункты составляли не более половины суммарной трудоемкости комплексного задания