**24.6 Практическая работа**

Цели практической работы

Научиться:

* писать программы в объектно-ориентированном стиле, выделять отдельные модели и инициализировать классы, работать с экземплярами класса;
* описывать методы класса, использовать аргумент self;
* использовать конструктор init для инициализации класса;
* работать с несколькими классами, прописывать их взаимодействие друг с другом;
* создавать и подключать свой собственный модуль.

Что входит в работу

Задача 1. Драка.

Задача 2. Студенты.

Задача 3. Круг.

Задача 4. Отцы, матери и дети.

Задача 5. Весёлая ферма 2.

Задача 6. Магия.

Задача 7. Совместное проживание.

Задача 8. Блек-джек.

Задача 9. Крестики-нолики.

Задача 1. Драка

**Что нужно сделать**

Вы работаете в команде разработчиков мобильной игры, и вам досталась такая часть от ТЗ заказчика:

Есть два юнита, каждый из них называется «Воин». Каждому устанавливается здоровье в 100 очков. Они бьют друг друга в случайном порядке. Тот, кто бьёт, здоровье не теряет. У того, кого бьют, оно уменьшается на 20 очков от одного удара. После каждого удара надо выводить сообщение, какой юнит атаковал и сколько у противника осталось здоровья. Как только у кого-то заканчивается ресурс здоровья, программа завершается сообщением о том, кто одержал победу.

Реализуйте такую программу.

**Что оценивается**

* Результат вычислений корректен.
* Модели реализованы в стиле ООП, основной функционал описан в методах классов и в отдельных функциях.
* Сообщения о процессе получения результата осмысленны и понятны для пользователя.
* Переменные, функции и собственные методы классов имеют значащие имена, не a, b, c, d.

Задача 2. Студенты

**Что нужно сделать**

Реализуйте модель с именем Student, содержащую поля: «ФИ», «Номер группы», «Успеваемость» (список из пяти элементов). Затем создайте список из десяти студентов (данные о студентах можете придумать свои или запросить их у пользователя) и отсортируйте его по возрастанию среднего балла. Выведите результат на экран.

**Что оценивается**

* Результат вычислений корректен.
* Input содержит корректные приглашения для ввода.
* Модели реализованы в стиле ООП, основной функционал описан в методах классов и в отдельных функциях.
* Сообщения о процессе получения результата осмысленны и понятны для пользователя.
* Переменные, функции и собственные методы классов имеют значащие имена, не a, b, c, d.

Задача 3. Круг

**Что нужно сделать**

На координатной плоскости рисуются круги, у каждого круга следующие параметры: координаты X и Y центра круга и значение R ― радиус круга. По умолчанию центр находится в (0, 0), а радиус равен 1.

Реализуйте класс «Круг», который инициализируется по этим параметрам. Круг также может:

1. Находить и возвращать свою площадь.
2. Находить и возвращать свой периметр.
3. Увеличиваться в K раз.
4. Определять, пересекается ли он с другой окружностью.

**Что оценивается**

* Результат вычислений корректен.
* Модели реализованы в стиле ООП, основной функционал описан в методах классов и в отдельных функциях.
* Переменные, функции и собственные методы классов имеют значащие имена, не a, b, c, d.

Задача 4. Отцы, матери и дети

**Что нужно сделать**

Реализуйте два класса: «Родитель» и «Ребёнок». У родителя есть:

1. Имя.
2. Возраст.
3. Список детей.

И он может:

1. Сообщить информацию о себе.
2. Успокоить ребёнка.
3. Покормить ребёнка.

У ребёнка есть:

1. Имя.
2. Возраст (должен быть меньше возраста родителя хотя бы на 16 лет).
3. Состояние спокойствия.
4. Состояние голода.

Реализация состояний на ваше усмотрение. Это может быть и простой «флаг», и словарь состояний, и что-нибудь ещё интереснее.

**Что оценивается**

* Результат вычислений корректен.
* Модели реализованы в стиле ООП, основной функционал описан в методах классов и в отдельных функциях.
* Переменные, функции и собственные методы классов имеют значащие имена, не a, b, c, d.

Задача 5. Весёлая ферма 2

**Что нужно сделать**

Мы продолжаем писать игру «Весёлая ферма», и теперь необходимо её немного модернизировать. Всё-таки кому-то нужно собирать урожай, и для этого нам понадобится садовник, у которого есть:

1. Имя.
2. Грядка с растением, за которым он ухаживает (в нашем случае пока только грядка с картошкой).

И он может:

1. Ухаживать за грядкой.
2. Собирать с неё урожай (количество картошки ― пустой список).

Модернизируйте программу, используя новый класс «Садовник». На всякий случай даём описание картошки и грядки:

У картошки есть её номер в грядке, а также стадия зрелости. Она может предоставлять информацию о своей зрелости и расти. Всего у картошки может быть четыре стадии зрелости.

Грядка с картошкой содержит список картошки, которая на ней растёт, и может, собственно, взращивать всю эту картошку, а также предоставлять информацию о зрелости всей картошки на своей территории.

Проверьте работу программы, создав грядку из пяти картошек и отдав эту грядку садовнику. Пусть поухаживает за грядкой и соберёт урожай (а может быть даже и не один).

**Что оценивается**

* Результат вычислений корректен.
* Input содержит корректные приглашения для ввода.
* Модели реализованы в стиле ООП, основной функционал описан в методах классов и в отдельных функциях.
* Сообщения о процессе получения результата осмысленны и понятны для пользователя.
* Переменные, функции и собственные методы классов имеют значащие имена, не a, b, c, d.

Задача 6. Магия

**Что нужно сделать**

Для одной игры необходимо реализовать механику магии, где при соединении двух элементов получается новый. У нас есть четыре базовых элемента: «Вода», «Воздух», «Огонь», «Земля». Из них как раз и получаются новые: «Шторм», «Пар», «Грязь», «Молния», «Пыль», «Лава».

Вот таблица преобразований:

* Вода + Воздух = Шторм
* Вода + Огонь = Пар
* Вода + Земля = Грязь
* Воздух + Огонь = Молния
* Воздух + Земля = Пыль
* Огонь + Земля = Лава

Напишите программу, которая реализует все эти элементы. Каждый элемент необходимо организовать как отдельный класс. Если результат не определён, то возвращается None.

Примечание: сложение объектов можно реализовывать через магический метод \_\_add\_\_, вот пример использования:

class Example\_1:  
    def \_\_add\_\_(self, other):  
        return Example\_2()  
  
  
class Example\_2:  
    answer = 'сложили два класса и вывели'  
  
  
a = Example\_1()  
b = Example\_2()  
c = a + b  
print(c.answer)

**Дополнительно:** придумайте свой элемент (или элементы), а также реализуйте его взаимодействие с остальными элементами.

**Что оценивается**

* Результат вычислений корректен.
* Модели реализованы в стиле ООП, основной функционал описан в методах классов и в отдельных функциях.
* Переменные, функции и собственные методы классов имеют значащие имена, не a, b, c, d.

Задача 7. Совместное проживание

**Что нужно сделать**

Чтобы понять, стоит ли ему жить с кем-то или всё же лучше остаться в гордом одиночестве, Артём решил провести довольно необычное исследование. Для этого он реализовал модель человека и модель дома.

Человек может:

1. Есть (+ сытость, − еда).
2. Работать (− сытость, + деньги).
3. Играть (− сытость).
4. Ходить в магазин за едой (+ еда, − деньги).

У человека есть имя, степень сытости (изначально 50) и дом.

В доме есть холодильник с едой (изначально 50 еды) и тумбочка с деньгами (изначально 0 денег).

Если сытость человека становится меньше нуля, то человек умирает.

Логика действий человека определяется следующим образом:

1. Генерируется число кубика от 1 до 6.
2. Если сытость < 20, то поесть.
3. Иначе, если еды в доме < 10, то сходить в магазин.
4. Иначе, если денег в доме < 50, то работать.
5. Иначе, если кубик равен 1, то работать.
6. Иначе, если кубик равен 2, то поесть.
7. Иначе играть.

По такой логике эксперимента человеку надо прожить 365 дней.

Реализуйте такую программу и создайте двух людей, живущих в одном доме. Проверьте работу программы несколько раз. Надеемся, эти люди живы...

**Что оценивается**

* Результат вычислений корректен.
* Модели реализованы в стиле ООП, основной функционал описан в методах классов и в отдельных функциях.
* Сообщения о процессе получения результата осмысленны и понятны для пользователя.
* Переменные, функции и собственные методы классов имеют значащие имена, не a, b, c, d.

Задача 8. Блек-джек

**Что нужно сделать**

Костя так и не смог завязать с азартными играми. Но перед тем как в очередной раз всё проиграть, он решил как следует подготовиться. И написать программу, на которой он будет тренироваться играть в блек-джек.

Блек-джек также известен как 21. Суть игры проста: нужно или набрать ровно 21 очко, или набрать очков больше, чем в руках у дилера, но ни в коем случае не больше 21. Если игрок собирает больше 21, он «сгорает». В случае ничьей игрок и дилер остаются при своих.

Карты имеют такие «ценовые» значения:

* от двойки до десятки — от 2 до 10 соответственно;
* у туза — 1 или 11 (11 пока общая сумма не больше 21, далее 1);
* у «картинок» (король, дама, валет) — 10.

Напишите программу, которая вначале случайным образом выдаёт пользователю и компьютеру по две карты и затем запрашивает у пользователя действие: взять карту или остановиться. На экран должна выдаваться информация о руке пользователя. После того как игрок останавливается, выведите на экран победителя.

Представление карты реализуйте с помощью класса.

**Дополнительно:** сделайте так, чтобы карты не могли повторяться.

Ваши классы в этой задаче могут выглядеть так:

class Card:  
    #  Карта, у которой есть значения  
    #   - масть  
    #   - ранг/принадлежность 2, 3, 4, 5, 6, 7 и так далее  
  
  
class Deck:  
    #  Колода создаёт у себя объекты карт  
  
  
class Player:  
    #  Игрок, у которого есть имя и какие-то карты на руках

**Что оценивается**

* Результат вычислений корректен.
* Модели реализованы в стиле ООП, основной функционал описан в методах классов и в отдельных функциях.
* Сообщения о процессе получения результата осмысленны и понятны для пользователя.
* Переменные, функции и собственные методы классов имеют значащие имена, не a, b, c, d.

Задача 9. Крестики-нолики

**Что нужно сделать**

Напишите программу, которая реализует игру «Крестики-нолики». Да, это всё условие задачи.

Ваши классы в этой задаче могут выглядеть так:

class Cell:  
    #  Клетка, у которой есть значения  
    #   - занята она или нет  
    #   - номер клетки  
  
  
class Board:  
    #  Класс поля, который создаёт у себя экземпляры клетки  
  
  
class Player:  
    #  У игрока может быть  
    #   - имя  
    #   - на какую клетку ходит

**Что оценивается**

* Результат вычислений корректен.
* Модели реализованы в стиле ООП, основной функционал описан в методах классов и в отдельных функциях.
* Сообщения о процессе получения результата осмысленны и понятны для пользователя.
* Переменные, функции и собственные методы классов имеют значащие имена, не a, b, c, d.

Что оценивается в практической работе

* Работа сдана через GitLab.
* Структура папок и файлов репозитория соответствует репозиторию python\_basic.
* Все задачи выполнены в соответствующих папках и файлах main.py.
* Описания коммитов осмысленны и понятны: 111, done, «я сделалъ» — неверно, added m15 homework, 14.3 fix: variables naming — верно.
* Использованы именованные индексы, не просто i (подробнее в видео 7.2).
* Использованы правильные числа, без дополнительных действий со стороны пользователя, без +1 (подробнее об этом в видео 7.4).
* Правильно оформлен input, без пустого приветствия для ввода (подробнее об этом в видео 2.3).
* Переменные и функции имеют значащие имена, не только a, b, c, d (подробнее об этом в видео 2.3).
* Есть пробелы после запятых и при бинарных операциях.
* Нет пробелов после имён функций и перед скобками: print (),input () — неверно, print() — верно.
* Правильно оформлены блоки if-elif-else, циклы и функции, отступы одинаковы во всех блоках одного уровня.
* Все входные и выходные файлы называются так, как указано в задачах.
* Работа с файлами осуществляется с помощью контекстного менеджера with.
* Для обработки исключений используются блоки try-except.
* Модели реализованы в стиле ООП, основной функционал описан в методах классов и в отдельных функциях.
* Если классы вынесены в отдельный модуль, то импортируются определённые классы (запись вида from garden import \* считается плохим тоном).

Рекомендации

* Арифметические операции [PEP8](https://docs.python.org/3.7/reference/expressions.html#operator-precedence) остаются в приоритете. Необходимо вводить and, or.
* Руководство по стилю Python [PEP8](https://www.python.org/dev/peps/pep-0008/) на английском языке.
* Руководство по стилю Python [PEP8](https://pythonworld.ru/osnovy/pep-8-rukovodstvo-po-napisaniyu-koda-na-python.htmlhttps:/pep8.ru/doc/pep8/) на русском языке.
* [Список встроенных функций.](https://docs.python.org/3.7/library/functions.html" \t "_blank)