**25.6 Практическая работа**

Цели практической работы

* Отработать применение принципа инкапсуляции в Python:
  + защита атрибутов класса от случайных внешних изменений, создание приватных атрибутов и методов класса;
  + сеттеры и геттеры.
* Отработать применение принципа наследования в Python:
  + создание нового класса на основе уже существующего, рассмотрение понятия базового (родительского) и дочернего класса;
  + дополнение дочернего класса новыми атрибутами, функция super.
* Отработать применение принципа полиморфизма в Python: изменение функционала, который наследуется от базового класса.
* Отработать написание документации для классов и методов, улучшить стиль программирования.

Что входит в работу

Задача 1. Налоги.

Задача 2. Карма.

Задача 3. Свой словарь.

Задача 4. Компания.

Задача 5. А-а-автомобиль!

Задача 6. Совместное проживание 2.

Задача 7. Стек.

Задача 1. Налоги

**Что нужно сделать**

Реализуйте иерархию классов, описывающих имущество налогоплательщиков. Она должна состоять из базового класса Property и производных от него классов Apartment, Car и CountryHouse.

Базовый класс должен иметь атрибут worth (стоимость), который передаётся в конструктор, и метод расчёта налога, переопределённый в каждом из производных классов. Налог на квартиру вычисляется как 1/1000 её стоимости, на машину — 1/200, на дачу — 1/500.

Каждый дочерний класс должен иметь конструктор с одним параметром, передающий свой параметр конструктору базового класса.

Разработайте интерфейс программы. Пусть она запрашивает у пользователя количество его денег и стоимость имущества, а затем выдаёт налог на соответствующее имущество и показывает, сколько денег ему не хватает (если это так).

**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.
* При написании классов соблюдаются основные принципы ООП: инкапсуляция, наследование и полиморфизм.
  + Для получения и установки значений у приватных атрибутов используются сеттеры и геттеры.
  + Для создания нового класса на основе уже существующего используется наследование.
* Сообщения о процессе получения результата осмыслены и понятны для пользователя.
* Переменные, функции и собственные методы классов имеют значащие имена, а не a, b, c, d.
* Классы и методы/функции имеют прописанную документацию.

Задача 2. Карма

**Что нужно сделать**

Один буддист-программист решил создать свой симулятор жизни, в котором нужно набрать 500 очков кармы (это константа), чтобы достичь просветления.

Каждый день вызывается специальная функция one\_day(), которая возвращает количество кармы от 1 до 7 и может с вероятностью 1 к 10 выкинуть одно из исключений:

* KillError;
* DrunkError;
* CarCrashError;
* GluttonyError;
* DepressionError.

Напишите такую программу. Функцию оберните в бесконечный цикл, выход из которого возможен только при накоплении кармы до уровня константы. Исключения обработайте и запишите в отдельный лог karma.log.

**Что оценивается**

* Результат вычислений корректен.
* Модели реализованы в стиле ООП, основной функционал описан в методах классов и в отдельных функциях.
* При написании классов соблюдаются основные принципы ООП: инкапсуляция, наследование и полиморфизм.
  + Для получения и установки значений у приватных атрибутов используются сеттеры и геттеры.
  + Для создания нового класса на основе уже существующего используется наследование.
* Сообщения о процессе получения результата осмыслены и понятны для пользователя.
* Переменные, функции и собственные методы классов имеют значащие имена, а не a, b, c, d.
* Классы и методы/функции имеют прописанную документацию.
* Названия используемых файлов соответствуют тем, которые написаны в задаче.

Задача 3. Свой словарь

**Что нужно сделать**

В силу обстоятельств Васе постоянно приходится работать со словарями и их данными. В том числе и с методом get, который по умолчанию возвращает None, если такого ключа в словаре нет. Однако Васю это не устраивает: для нормальной работы ему нужно возвращать число 0.

Реализуйте класс MyDict, который будет вести себя точно так же, как и обычный словарь, за исключением того, что метод get по умолчанию будет возвращать не None, а число 0.

**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 осмыслены и понятны для пользователя.
* Переменные, функции и собственные методы классов имеют значащие имена, а не a, b, c, d.
* Классы и методы/функции имеют прописанную документацию.

Задача 4. Компания

**Что нужно сделать**

Реализуйте иерархию классов, описывающих служащих в компании. На самом верху иерархии — класс Person, который описывает человека именем, фамилией и возрастом. Все атрибуты этого класса являются приватными.

Далее идёт класс Employee и производные от него классы Manager, Agent и Worker.

Класс «Работник» должен иметь метод расчёта заработной платы, переопределённый в каждом из производных классов. Заработная плата Manager постоянна и равна 13000, заработная плата Agent определяется как оклад 5000 + 5% от объёма продаж, который хранится в специальном поле класса Agent, а заработная плата Worker определяется как 100 \* число отработанных часов, которое также хранится в отдельном поле.

В основной программе создайте список из девяти объектов: первые три — Manager, следующие три — Agent и последние три — Worker. Выведите на экран величину заработной платы всех девяти служащих.

**Что оценивается**

* Результат вычислений корректен.
* Модели реализованы в стиле ООП, основной функционал описан в методах классов и в отдельных функциях.
* При написании классов соблюдаются основные принципы ООП: инкапсуляция, наследование и полиморфизм.
  + Для получения и установки значений у приватных атрибутов используются сеттеры и геттеры.
  + Для создания нового класса на основе уже существующего используется наследование.
* Сообщения о процессе получения результата осмыслены и понятны для пользователя.
* Переменные, функции и собственные методы классов имеют значащие имена, а не a, b, c, d.
* Классы и методы/функции имеют прописанную документацию.

Задача 5. А-а-автомобиль!

**Что нужно сделать**

Автомобиль имеет координаты своего положения и угол, описывающий направление движения. Он может быть изначально поставлен в любую точку с любым направлением (конструктор), может проехать в выбранном направлении определённое расстояние и может повернуть, то есть изменить текущее направление на любое другое (передаём привет математике и формулам).

Реализуйте класс автомобиля, а также класс, который будет описывать автобус. У автобуса, кроме того, что имеется у автомобиля, должны быть поля, содержащие число пассажиров и количество полученных денег, изначально равное нулю. Также должны быть методы «войти» и «выйти», изменяющие число пассажиров. Наконец, метод move должен быть переопределён, чтобы увеличивать количество денег в соответствии с количеством пассажиров и пройденным расстоянием.

**Что оценивается**

* Результат вычислений корректен.
* Модели реализованы в стиле ООП, основной функционал описан в методах классов и в отдельных функциях.
* При написании классов соблюдаются основные принципы ООП: инкапсуляция, наследование и полиморфизм.
  + Для получения и установки значений у приватных атрибутов используются сеттеры и геттеры.
  + Для создания нового класса на основе уже существующего используется наследование.
* Сообщения о процессе получения результата осмыслены и понятны для пользователя.
* Переменные, функции и собственные методы классов имеют значащие имена, а не a, b, c, d.
* Классы и методы/функции имеют прописанную документацию.

Задача 6. Совместное проживание 2

**Что нужно сделать**

Артём увлёкся предыдущим экспериментом и решил расширить его, создав целую семью из Мужа, Жены и Кота. Условия эксперимента следующие.

Каждый день участники жизни могут делать только одно действие. Все вместе они должны прожить год и не умереть.

Муж может:

1. есть;
2. играть;
3. ходить на работу.

Жена может:

1. есть;
2. покупать продукты;
3. покупать шубу;
4. убираться в доме.

Кот может:

1. есть;
2. спать;
3. драть обои.

Все они живут в одном доме, дом характеризуется:

1. количеством денег в тумбочке (вначале 100);
2. количеством еды в холодильнике (вначале 50);
3. едой для кота (вначале 30);
4. количеством грязи (вначале 0).

У людей есть имя, степень сытости (вначале 30) и степень счастья (вначале 100). Все люди могут гладить кота (+5 к счастью).

У кота есть имя и степень сытости (вначале 30).

Любое действие (в том числе и кота), кроме «есть», приводит к уменьшению степени сытости на 10 пунктов.

Взрослые едят максимум по 30 единиц еды, степень сытости растёт на один пункт за один пункт еды.

Кот ест максимум по 10 единиц еды, степень сытости растёт на два пункта за один пункт еды.

Степень сытости не должна падать ниже нуля, иначе человек или кот умрёт от голода.

Деньги в тумбочку добавляет муж после работы — сразу 150 единиц.

Еда стоит 10 денег за 10 единиц еды. Шуба стоит 350 единиц.

Еда для кота тоже покупается — за 10 денег 10 еды.

Грязь добавляется каждый день по пять пунктов, за одну уборку жена может убирать до 100 единиц грязи.

Если кот дерёт обои, то грязи тоже становится больше на пять пунктов.

Если в доме грязи больше 90, у людей падает степень счастья каждый день на 10 пунктов.

Степень счастья растёт: у мужа от игры в компьютер (на 20), у жены от покупки шубы (на 60, но шуба дорогая).

Степень счастья не должна падать ниже 10, иначе человек умирает от депрессии.

Реализуйте такую программу. Подведите итоги жизни за год — сколько было заработано денег, сколько съедено еды, сколько куплено шуб.

**Дополнительно:** добавьте ещё ребёнка и несколько котов.

**Что оценивается**

* Результат вычислений корректен.
* Модели реализованы в стиле ООП, основной функционал описан в методах классов и в отдельных функциях.
* При написании классов соблюдаются основные принципы ООП: инкапсуляция, наследование и полиморфизм.
  + Для получения и установки значений у приватных атрибутов используются сеттеры и геттеры.
  + Для создания нового класса на основе уже существующего используется наследование.
* Сообщения о процессе получения результата осмыслены и понятны для пользователя.
* Переменные, функции и собственные методы классов имеют значащие имена, а не a, b, c, d.
* Классы и методы/функции имеют прописанную документацию.

Задача 7. Стек

**Что нужно сделать**

Мы уже говорили, что в программировании нередко необходимо создавать свои собственные структуры данных на основе уже существующих. Одной из таких «базовых» структур является стек.

Стек — это абстрактный тип данных, представляющий собой список элементов, организованных по принципу LIFO (англ. last in — first out, «последним пришёл — первым вышел»).

Простой пример: стек из книг на столе. Единственной книгой, чья обложка видна, является самая верхняя. Чтобы получить доступ к, например, третьей снизу книге, нам нужно убрать все книги, лежащие сверху, одну за другой.

Напишите класс, который реализует стек и его возможности (достаточно будет добавления и удаления элемента).

После этого напишите ещё один класс — «Менеджер задач». В менеджере задач можно выполнить команду «новая задача», в которую передаётся сама задача (str) и её приоритет (int). Сам менеджер работает на основе стека (не наследование!). При выводе менеджера в консоль все задачи должны быть отсортированы по приоритету: чем меньше число, тем выше задача.

Вот пример основной программы:

manager = TaskManager()  
manager.new\_task("сделать уборку", 4)  
manager.new\_task("помыть посуду", 4)  
manager.new\_task("отдохнуть", 1)  
manager.new\_task("поесть", 2)  
manager.new\_task("сдать дз", 2)  
print(manager)

Результат:

1 отдохнуть

2 поесть; сдать дз

4 сделать уборку; помыть посуду

**Дополнительно**: реализуйте также удаление задач и подумайте, что делать с дубликатами.

**Что оценивается**

* Результат вычислений корректен.
* Модели реализованы в стиле ООП, основной функционал описан в методах классов и в отдельных функциях.
* При написании классов соблюдаются основные принципы ООП: инкапсуляция, наследование и полиморфизм.
  + Для получения и установки значений у приватных атрибутов используются сеттеры и геттеры.
  + Для создания нового класса на основе уже существующего используется наследование.
* Формат вывода соответствует примеру.
* Переменные, функции и собственные методы классов имеют значащие имена, а не a, b, c, d.
* Классы и методы/функции имеют прописанную документацию.

Что оценивается в практической работе

* Задание сдано через GitLab.
* Структура папок и файлов репозитория соответствует репозиторию python\_basic.
* Все задачи выполнены в соответствующих папках и файлах main.py.
* Описания коммитов осмыслены и понятны: 111, done, «я сделалъ» — неверно; added m15 homework, 14.3 fix: variables naming — верно.
* Использованы именованные индексы, а 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, а не только a, b, c, d (подробнее об этом в видео 2.3).
* Присутствуют пробелы после запятых и при бинарных операциях.
* Отсутствуют пробелы 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 заданиях.
* 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.
* При написании классов соблюдаются основные принципы ООП: инкапсуляция, наследование и полиморфизм.
  + Для получения и установки значений у приватных атрибутов используются сеттеры и геттеры.
  + Для создания нового класса на основе уже существующего используется наследование.
* Если классы вынесены в отдельный модуль, то импортируются определённые классы (запись вида from garden import \* считается плохим тоном).
* Классы и методы/функции имеют прописанную документацию.

Советы и 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.html)на русском языке.
* [Список встроенных функций.](https://docs.python.org/3.7/library/functions.html)

Как отправить работу на проверку

Чтобы выполнить практическую работу, обновите репозиторий dpo\_python\_basic на своём компьютере при помощи IDE PyCharm. Задачи находятся в папке Module25.

Сдайте практические работы этого модуля через систему контроля версий Git сервиса Skillbox Gitlab. В материалах с практической работой напишите «Сделано» и прикрепите ссылку на репозиторий. Ссылки на реплит оставлять не нужно.