Задачи на ООП. Блок 2.

1. Определите классы «Студент» и «Курс» для студентов и курсов, которые они изучают. При этом, определите следующие атрибуты и поведение классов:

Атрибуты класса «Студент»: имя, идентификатор, список пройденных курсов. Поведение: запись на курс, удаление курса, просмотр пройденных курсов.
Класс "Курс": Атрибуты: название курса, инструктор, макс. кол-во студентов. Поведение: добавление учеников, удаление учеников, просмотр зачисленных учеников.

Далее, создайте объекты для классов и протестируйте систему.

2. Определите класс с именем myString, наследующий класс str, и позволяющий наделять строки методами append () и pop (), выполняющими те же операции, что и у класса list.
3. Реализуйте простой пример перегрузки оператора, создав класс "Point", который перегружает оператор +, чтобы сложить две точки (т.е. их координаты) вместе.
4. Создайте базовый класс под названием «Animal» и два подкласса «Dog» и «Cat». Добавьте методы и атрибуты, специфичные для каждого подкласса. Создайте функцию, которая принимает объект animal в качестве входных данных и вызывает его метод “sound”. Проверьте его как с объектом "dog", так и с объектом "cat".
5. Создайте три класса: “Person”, “Employee” и “Student.”. Используйте множественное наследование для создания класса «PersonInfo», который наследует как от «Person» , так и от «Employee» и «Student». Добавьте атрибуты и методы, специфичные для каждого класса. Обратите внимание на конструирование метода инициализации класса «PersonInfo».
6. Создайте класс Soda (для определения типа газированной воды), принимающий 1 аргумент при инициализации (отвечающий за добавку к выбираемому лимонаду). В этом классе реализуйте метод show_my_drink(), выводящий на печать «Газировка и ДОБАВКА» в случае наличия добавки, а иначе отобразится следующая фраза: «Обычная газировка». При решении задания нужно дополнительно проверить тип передаваемого аргумента: принимается только строка.
7. Создайте классы «Author» и «Book», демонстрирующие композицию между классами. Класс Book должен иметь атрибуты: Title, Author (Полное имя), Price. Определите конструктор, используемый для инициализации атрибутов метода со значениями, введенными пользователем. Задайте метод View () для отображения информации для текущей книги. Написать программу для тестирования класса Book.
8. Создайте классы «Department» и «Employee». Используйте агрегацию для представления принадлежности сотрудника отделу. Реализуйте

методы для добавления сотрудников в отдел и расчета средней заработной платы сотрудников в отделе.

9. Создайте класс Nikola, принимающий при инициализации 2 параметра: имя и возраст. Не важно, какое имя передаст пользователь при создании экземпляра, оно всегда должно быть “Николай”. В частности - если пользователя на самом деле зовут Николаем, то с именем ничего не произойдет, а если его зовут, например, Максим, то должно выводится сообщение “Я не Максим, а Николай”. Более того, никаких других атрибутов и методов у экземпляра не может быть добавлено, даже если кто-то и вздумает так поступить (т.е. если некий пользователь решит прибавить к экземпляру свойство «отчество» или метод «приветствие», то ничего у него не получится). Для ограничения количества наборов свойств и методов в экземпляре применяется специальный магический атрибут__slots__.
10. Создайте класс KgToPounds с параметром kg, куда передается определенное количество килограмм, а с помощью метода to_pounds() они переводятся в фунты. Чтобы закрыть доступ к переменной “kg”, реализуйте методы set_kg() - для задания нового значения килограммов, get_kg() - для вывода текущего значения кг. В результате нам нужно теперь использовать эти 2 метода для задания и вывода значений. Устраните это неудобство с использованием функции property() или свойств-декоратора @property и метода-сеттера.
11. Пусть есть Алфавит, характеристиками которого являются «Язык» и «Список букв». Для Алфавита можно: напечатать все буквы алфавита, посчитать количество букв. Пусть также есть Английский алфавит, который обладает следующими свойствами: «Язык», «Список букв», «Количество букв». Для Английского алфавита можно: посчитать количество букв, определить, относится ли буква к английскому алфавиту, получить пример текста на английском языке. Создайте класс Alphabet. Создайте метод __init__(), внутри которого будут определены два динамических свойства: 1) lang - язык и 2) letters - список букв. Начальные значения свойств берутся из входных параметров метода. Создайте метод print(), который выведет в консоль буквы алфавита. Создайте метод letters_num(), который вернет количество букв в алфавите.

Далее, создайте класс EngAlphabet путем наследования от класса Alphabet. Создайте метод __init__(), внутри которого будет вызываться родительский метод __init__(). В качестве параметров ему будут передаваться обозначение языка (например, 'En') и строка, состоящая из всех букв алфавита (можно воспользоваться свойством ascii_uppercase из модуля string). Добавьте приватное статическое свойство __letters_num, которое будет хранить количество букв в алфавите.

Создайте метод is_en_letter(), который будет принимать букву в качестве параметра и определять, относится ли эта буква к английскому алфавиту. Переопределите метод letters_num() - пусть в текущем классе он будет возвращать значение свойства __letters_num.

Создайте статический метод example(), который будет возвращать пример текста на английском языке.

Протестируйте решение. Для этого создайте объект класса EngAlphabet, напечатайте буквы алфавита для этого объекта, выведите количество букв в алфавите, проверьте, относится ли буква F к английскому алфавиту, проверьте, относится ли буква Щ к английскому алфавиту, выведите пример текста на английском языке.

12. Рассмотрим следующую предметную область. Пусть в поле есть «Помидор» со следующими характеристиками: Индекс, Стадия зрелости (стадии: Отсутствует, Цветение, Зеленый, Красный). Помидор может расти (переходить на следующую стадию созревания), предоставлять информацию о своей зрелости. Есть также «Куст с помидорами», который содержит список томатов, которые на нем растут и может расти вместе с томатами, предоставлять информацию о зрелости всех томатов, предоставлять урожай. За помидорами ухаживает «Садовник», который имеет: имя, растение, за которым он ухаживает. Он может ухаживать за растением, убирать с него урожай.

Создайте класс «Tomato» со статическим свойством states, которое будет содержать все стадии созревания помидора.

Создайте метод __init__(), внутри которого будут определены два динамических protected свойства: 1) _index - передается параметром и 2) _state - принимает первое значение из словаря states. Создайте метод grow(), который будет переводить томат на следующую стадию созревания и метод is_ripe(), который будет проверять, что томат созрел (достиг последней стадии созревания).

Далее, создайте класс «TomatoBush» с методом __init__(), который будет принимать в качестве параметра количество томатов и на его основе будет создавать список объектов класса Tomato.

Данный список будет храниться внутри динамического свойства tomatoes. Создайте метод grow_all(), который будет переводить все объекты из списка томатов на следующий этап созревания.

Создайте метод all_are_ripe(), который будет возвращать True, если все томаты из списка стали спелыми.

Создайте метод give_away_all(), который будет чистить список томатов после сбора урожая. Создайте класс «Gardener» с методом инициализаци __init__(), внутри которого будут определены два динамических свойства: 1) name - передается параметром, является публичным и 2) _plant - принимает объект класса TomatoBush, (является protected).

Создайте метод work(), который заставляет садовника работать, что позволяет растению становиться более зрелым.

Создайте метод harvest(), который проверяет, все ли плоды созрели. Если все плоды созрели - садовник собирает урожай. Если нет - метод печатает предупреждение.

Создайте статический метод knowledge_base(), который выведет в консоль справку по садоводству.

Протестируйте решение. Для этого вызовите справку по садоводству, создайте объекты классов TomatoBush и Gardener. Используя объект

класса Gardener, поухаживайте за кустом с помидорами. Попробуйте собрать урожай. Если томаты еще не дозрели, продолжайте ухаживать за ними. Соберите урожай.

13. Определите класс под названием Individual. Добавьте метод инициализации, который инициализирует атрибут экземпляра self.character_name. Добавьте метод доступа к классу get_character_name (), возвращающий self.character_name. Создайте экземпляр класса и назначьте его переменной individual1. Экземпляру этого класса при инициализации должен быть назначено character_name «Buster». Создайте другой экземпляр, который должен быть назначен переменной individual2. Установите для него имя «Тобиас». Напечатайте имена individual1 и individual2 на экране, используя соответствующий метод.

Далее, для класса Individual при инициализации присвойте атрибуту экземпляра self.happy значение True. Это должно быть сделано по умолчанию (то есть для этого не нужно передавать параметр при создании экземпляра).

Создайте метод is_happy, возвращающий статус self.happy. Создайте метод модификации с именем switch_mood (), который изменяет self.happy с True на False (и наоборот).

Создайте метод, называемый speak(), который возвращает «Привет, я [self.name]» или «Уходи!», в зависимости от того, установлен ли self.happy в True или False соответственно.

Создайте объект individual3 с именем, инициализированным как «Lucille». Напишите некоторый код, чтобы протестировать эти методы/атрибуты для Buster и Tobias.

Добавьте атрибут класса с именем self.Counter, записывающий количество созданных отдельных экземпляров. Это значение должно быть увеличено методом класса с именем AddOne (). Таким образом, можно отслеживать общее количество людей. Текущее значение счетчика должно быть назначено переменной self.id при инсталляции. Напишите методы __str __ и __ repr __, чтобы дать более понятное представление каждого экземпляра Individual. Они должен вернуть: individual: [self.id self.character_name].

Напишите дополнительный код, чтобы убедиться, что класс работает должным образом.

Создайте популяцию людей из сериала «звездные войны». В списке свойств должны быть категории: Name, Height, Mass, Homeworld, Species. В нашем классе Individual уже есть атрибут для хранения имен. Добавьте атрибуты self.height, self.mass и self.homeworld. Они должны быть установлены при создании отдельного экземпляра объекта. Создайте методы доступа для получения значений добавленных атрибутов.

Свойство species может принимать значение droid (робот) и living (;bdst ceotcndf). Они будут иметь несколько разные свойства, поэтому создайте подклассы класса Individual под названием Droid и Biological.

Добавьте атрибут вида species в классы Droid и Biological. Это должно быть сделано при создании экземпляра. Кроме того, добавьте метод

доступа для возврата значения species. Убедитесь, что это работает должным образом.

Создайте экземпляры droid или biological класса, используя данные из списка персонажей фильма. Эти вновь созданные объекты должны храниться в классе Population с именем списка population. Напишите некоторый код, чтобы проверить, что это сработало.

Переопределите метод speak в классе Droid , чтобы вернуть «Beep Beep Beep». Проверьте, что это работает.

Добавьте метод get_bmi () к классу Biological, который возвращает индекс массы тела биологического объекта. (Индекс массы тела - это простой расчет с использованием роста и веса человека. Формула BMI = mass / (height * height) (с массой в килограммах и ростом в метрах). Переберите список population, определив экземпляры биологического класса (функция isinstance () может помочь вам в этом) и запишите их значения индекса массы тела. Определите экземпляр с самым высоким индексом массы тела.

14. Строки в Питоне сравниваются на основании значений символов.

Т.е. если мы захотим выяснить, что больше: «Apple» или «Яблоко», – то «Яблоко» окажется бОльшим. Это так потому, что английская буква «A» имеет кодовое значение 65 (берется из таблицы кодировки), а русская буква «Я» – 1071 (с помощью функции ord() это можно выяснить).

Задание состоит в том, чтобы сравнивать строки по количеству входящих в них символов.

Для этого нужно создать класс RealString и реализовать соотвествующий инструментарий. Сравнивать между собой можно как объекты класса, так и обычные строки с экземплярами класса RealString.

Для воплощения задуманного понадобится только 3 метода внутри класса (включая __init__()).

Вообще-то для создания такого класса понадобится 4 метода, так как в Питоне реализованы сравнения >, <, >=, <=. Это значит, что если имеется сравнение «больше», то автоматом появится возможность осуществлять сравнение «меньше».

Чтобы сделать класс с тремя методами, требуется воспользоваться декоратором @total_ordering из модуля functools (упрощает реализацию сравнений. В этом случае потребуется лишь 2 дополняющих варианта сравнения.

15. Перед вами задача "Покупка дома". С помощью подхода ООП и средств Python в рамках данной задачи необходимо смоделировать следующую предметную область:

Есть Человек, характеристиками которого являются: Имя, Возраст, Наличие денег, Наличие собственного жилья.

Человек может: Предоставить информацию о себе, Заработать деньги, Купить дом.

Соответственно, есть Дом, к свойствам которого относятся: Площадь, Стоимость. Для Дома можно применить скидку на покупку.

Также есть Небольшой Типовой Дом, обязательной площадью 40м2.

Реализация задачи состоит из следующих этапов.

I. Реализация класса Human.

1. Создайте класс Human.
2. Определите для него два статических поля: default_name и default_age.
3. Создайте метод __init__(), который помимо self принимает еще два параметра: name и age. Для этих параметров задайте значения по умолчанию, используя свойства default_name и default_age. В методе __init__() определите четыре свойства: Публичные - name и age. Приватные - money и house.
4. Реализуйте справочный метод info(), который будет выводить поля name, age, house и money.
5. Реализуйте справочный статический метод default_info(), который будет выводить статические поля default_name и default_age.
6. Реализуйте приватный метод make_deal(), который будет отвечать за техническую реализацию покупки дома: уменьшать количество денег на счету и присваивать ссылку на только что купленный дом. В качестве аргументов данный метод принимает объект дома и его цену.
7. Реализуйте метод earn_money(), увеличивающий значение свойства money.
8. Реализуйте метод buy_house(), который будет проверять, что у человека достаточно денег для покупки, и совершать сделку. Если денег слишком мало - нужно вывести предупреждение в консоль. Параметры метода: ссылка на дом и размер скидки

II. Реализация класса House

1. Создайте класс House.
2. Создайте метод __init__() и определите внутри него два динамических свойства: _area и _price. Свои начальные значения они получают из параметров метода __init__().
3. Создайте метод final_price(), который принимает в качестве параметра размер скидки и возвращает цену с учетом данной скидки.

III. Реализация класса SmallHouse

1. Создайте класс SmallHouse, унаследовав его функционал от класса House.
2. Внутри класса SmallHouse переопределите метод __init__() так, чтобы он создавал объект с площадью 40м2.

IV. Тестирование решения

1. Вызовите справочный метод default_info() для класса Human.
2. Создайте объект класса Human.
3. Выведите справочную информацию о созданном объекте (вызовите метод info()).
4. Создайте объект класса SmallHouse.
5. Попробуйте купить созданный дом, убедитесь в получении предупреждения.
6. Поправьте финансовое положение объекта - вызовите метод earn_money().
7. Снова попробуйте купить дом.
8. Посмотрите, как изменилось состояние объекта класса Human.