Основы и методология программирования

Семинар 11

Классы

Парадигмы программирования

- Императивное (процедурное)
- Декларативное
- **Объектно-ориентированное**
- Функциональное
- Другие..

Объектно-ориентированное программирование

- Всё в программе построено на основе взаимодействующих **объектов**, являющихся **абстракциями** реального мира
- **Наследование**
- **Инкапсуляция**
- **Полиморфизм**

Классы, объекты и работа с ними

```
class Tree: # объявление класса
    #treeName = str()   # Объявление атрибута класса
    __treeName = str()   # Правильное объявление атрибута класса - private!
    # Нет особых спецификаторов для области видимости, вместо этого используется преффикс "__"

    def grow(self):   # Метод класса, self - обязательный формальный параметр для методов при их объявлении, при его отсутстви - статический метод
        print("I am a(an) %s. I am growing." % self.treeName)

    def getName(self):   # Метод для получения данных из объекта класса, "геттер" или "аксессор"
        return self.__treeName

    def setName(self, name):   # Метод для установки данных в объект класса, "сеттер" или "аксессор"
        self.__treeName = name

    def __init__(self):   # Конструктор класса
        self.__treeName = "Default Tree"
```

Классы, объекты и работа с ними

```
myOak = Tree()
myOak.treeName = "Oak"
myOak.__treeName = "Maple" # ???
myOak.grow()
myOak.size = 10  # Поля к классу можно добавлять "на лету" (но
ненужно)
print(myOak.size) # ???
print(myOak.__treeName) # ???
print(myOak.getName()) # ???
```

Задача 1

Создать класс Engine - Двигатель, с полями для хранения производителя, модели, объема бака и расхода топлива. Для полей определить методы доступа. Создать метод, возвращающий информацию в виде строки об объекте класса. В основной программе создать список, содержащий 5 объектов этого класса, затем рассчитать максимальное расстояние, которое может проехать каждый двигатель.

```
class Engine:
    def __init__(self):  # конструктор по умолчанию
        self.__producer = ""
        self.__model = ""
        self.__tankSize = 0
        self.__fuelConsumption = 0

    def __init__(self, *values):  # конструктор "общего" вида
        if (len(values) == 4):
            self.__producer, self.__model, self.__tankSize, self.__fuelConsumption = values
```

Задача 1

```
en1 = Engine("Volvo", "z35", "100", "15")
# ...
en5 = Engine("GMC", "gm11", "600", "70")

listOfEngines = [en1, en2, en3, en4, en5]
print(listOfEngines)  # что будет напечатано?

print(*map(lambda x: x.getInfo(), listOfEngines), sep="\n")

for engine in listOfEngines:
   print(round(engine.getTankSize() / engine.getConsumption(), 2), end=" ")
print()

# или тоже самое в одну строку
print(*map(lambda x: round(x.getTankSize() / x.getConsumption(), 2), listOfEngines))
```