Assignment 7
Object-Oriented Programming in Python

UAH, Departamento de Automática, ATC-SOL
http://atc1.aut.uah.es

Week 7

Objectives:

- Create single classes in Python
- Consume classes
- Create simple class hierarchies
- Understand overriding
- Handle trees with a module

Most of the following exercises have been collected from http://introtopython.org/classes.html#what

Classes

Exercise 1

Modeling a person is a classic exercise for people who are trying to learn how to write classes. We are all familiar with characteristics and behaviors of people, so it is a good exercise to try.

1. Define a `Person()` class.
2. In the `__init()__` function, define several attributes of a person. Good attributes to consider are name, age, place of birth, and anything else you like to know about the people in your life.
3. Write one method. This could be as simple as `introduce_yourself()`. This method would print out a statement such as, "Hello, my name is Eric."
4. You could also make a method such as `age_person()`. A simple version of this method would just add 1 to the person's age.
    - A more complicated version of this method would involve storing the person's birthdate rather than their age, and then calculating the age whenever the age is requested. But dealing with dates and times is not particularly easy if you've never done it in any other programming language before.

5. Create a person, set the attribute values appropriately, and print out information about the person.
6. Call your method on the person you created. Make sure your method executed properly; if the method does not print anything out directly, print something before and after calling the method to make sure it did what it was supposed to.

Exercise 2

Modeling a car is another classic exercise.

- Define a Car() class.
- In the `__init__()` function, define several attributes of a car. Some good attributes to consider are make (Subaru, Audi, Volvo...), model (Outback, allroad, C30), year, num_doors, owner, or any other aspect of a car you care to include in your class.
- Write one method. This could be something such as `describe_car()`. This method could print a series of statements that describe the car, using the information that is stored in the attributes. You could also write a method that adjusts the mileage of the car or tracks its position.
- Create a car object, and use your method.
- Create several car objects with different values for the attributes. Use your method on several of your cars.

Inheritance

Exercise 3

Start with your program from `Person` Class.

- Make a new class called `Student` that inherits from `Person`.
- Define some attributes that a student has, which other people don't have.
  - A student has a school they are associated with, a graduation year and other particular attributes.
- Create a Student object, and prove that you have used inheritance correctly.
  - Set some attribute values for the student, that are only coded in the Person class.
  - Set some attribute values for the student, that are only coded in the Student class.
  - Print the values for all of these attributes.

Exercise 4

Given the following `Rocket` class:

```
from math import sqrt

class Rocket():
    # Rocket simulates a rocket ship for a game,
    #  or a physics simulation.

    def __init__(self, x=0, y=0):
        # Each rocket has an (x,y) position.
        self.x = x
        self.y = y

    def move_rocket(self, x_increment=0, y_increment=1):
        # Move the rocket according to the paremeters given.
        #  Default behavior is to move the rocket up one unit.
        self.x += x_increment
        self.y += y_increment

    def get_distance(self, other_rocket):
        # Calculates the distance from this rocket to another rocket,
        #  and returns that value.
        distance = sqrt((self.x-other_rocket.x)**2+(self.y-other_rocket.y)**2)
        return distance
```

Perform the following tasks:

1. Write a `Shuttle` class, which is just a rocket that can be reused several times. To this end, extend the `Rocket` class and add the following constructor `__init()__(self,x=0, y=0, flights_completed=0)`
2. Create several shuttles and rockets, with random positions random number of flights completed (only for shuttles).
3. Add more attributes that are particular to shuttles such as maximum number of flights, capability of supporting spacewalks, and capability of docking with the ISS.
4. Add one more method to the class, that relates to shuttle behavior. This method could simply print a statement, such as "Docking with the ISS," for a `dock_ISS()` method.
5. Prove that your refinements work by creating a `Shuttle` object with these attributes, and then call your new method.

Class and modules

Exercise 5

Take your program from `Student` Class

1. Save your `Person` and `Student` classes in a separate file called `person.py`.

2. Save the code that uses these classes in four separate files.
   a) In the first file, use the `from module_name import ClassName` syntax to make your program run.
   b) In the second file, use the `import module_name` syntax.
   c) In the third file, use the `import module_name as different_local_module_name` syntax.
   d) In the fourth file, use the `import *` syntax.

Exercise 6

Take your program from `Car` Class

1. Save your `Car` class in a separate file called `car.py`.
2. Save the code that uses the car class into four separate files.
   a) In the first file, use the `from module_name import ClassName` syntax to make your program run.
   b) In the second file, use the `import module_name` syntax.
   c) In the third file, use the `import module_name as different_local_module_name` syntax.
   d) In the fourth file, use the `import *` syntax.

Trees

Download the files `node.py`, `tree.py` and `app.py` from `http://www.quesucede.com/page/show/id/python-3-tree-implementation`. This is an implementation of the tree data structure.

1. Read and understand the code in the three files, begining with `app.py`, which shows an example of how to use a tree.
2. Create a program that reprents the expression $(4 + 2^2) * 4 + 7$.
3. Search if there is a node with value 7 in the previous expression. Repeat the search with the value 34.
4. Model your family tree from your granfathers, search if "Jose" is a member of your family.
5. Which search algorithms are implemented? Locate the code that performs the search.