Practical 6
Modelling the World with Objects

Learning Objectives

1. Understand the main concepts in object-oriented programming and their value
2. Read and explain object-oriented code
3. Apply and create simple object-oriented Python code

Overview

In this practical you will write Python programs to implement the objects used as examples in the lecture. We will see how to create and manipulate objects, and how to group them in useful ways.

Tasks

1. Starting out with animals.py

We will keep our class definitions in separate python files. In your Practical 6 directory, edit animals.py with vim. Enter the class definition for Cat into animals.py.

If you try to run animals.py, what happens?

We need a driver program to test out our **Cat** class. Edit **testCat.py** and enter the following code:

```
from animals import Cat

garfield = Cat(‘Garfield’, ‘1/1/1978’, ‘Orange’, ‘Tabby’)

garfield.printit()

print(garfield)
```

Note the difference in the output of garfield.printit() and print(garfield).

2. More Animals!

Edit animals.py and add in the class definitions for **Dog** and **Bird**. Create some test data and code in **testAnimals.py** to test all three classes. Use testCat.py and the lecture notes as a starting point.

3. Even more animals!

Following the general outline for our programs, we will now create a program to read in animals from a file into a list of animal objects. The file will hold comma-separated values. Our algorithm will be:

1. Open file
2. Read data from file
3. Print animal list using printit() method

What we will need in our code is:

1. Import classes
2. Create variables
3. Open file
4. For each line in the file
    a. Create animal object to match first field in the entry
5. Print animal list

Have a look at **animals.csv** (below) to see how you can read in the file.

```
Dog,Dude,1/1/2011,Brown,Jack Russell
Cat,Oogie,1/1/2006,Grey,Fluffy
Bird,Big Bird,10/11/1969,Yellow,Canary
Cat,Garfield,1/1/1978,Orange,Tabby
```

Work through each part of the algorithm and test it out before moving on to the next step.

Extend the **animals.csv** file to include animals of your choice.

4. Building bank accounts

Type in the code from Version 3 of the Bank Accounts example in the lecture slides. The class definitions should be in **accounts.py** and the driver code in **testAccounts.py**. Test (run) the code to see that it works as shown in the lecture.

5. Simulating bank account transactions

We are going write some new code **banking.py** to allow the user to choose a transaction, account and optional amount to run transactions on **one account**. This will be in a loop that also allows for printing all the current balance. We are working towards something similar to the bucketlist builder…

1. Import classes
2. Create variables
    a. build bank account object (‘Everyday’, ‘007’, 3000)
3. Request transaction selection (withdrawal, deposit, interest), balance or exit e.g. W for Withdrawal, D for deposit, I for interest, B for balance and X for eXit

4. While not exit
    a. If ‘W’ – ask for amount then call withraw method
    b. Else if ‘D’ – ask for amount then call deposit method
    c. Else if ‘I’ – cal interest method
    d. Else if ‘B’ – print balance
    e. Request transaction selection

Submission

Create a README file for Prac9. Include the names and descriptions of all of your files from today.

All of your work for this week’s practical should be submitted via Blackboard using the link in the Week 6 unit materials. This should be done as a single "zipped" file.

Reflection

1. **Knowledge**: What is the difference between a class and an object?
2. **Comprehension**: What us the difference in the code used to define a class, and the code used to define an object?
3. **Application**: How would you write a new class to represent a rabbit?
4. **Analysis**: What is the difference between garfield.printit() and print(garfield)? Which would you use and why?
5. **Synthesis**: How would you modify the pet classes in animals.py to include a microchip number for cats and dogs?
6. **Evaluation**: Task 5 takes in user input to determine the transactions and amounts if required. What parts of this should have validation to make the code more robust? (less likely to crash if given the wrong input)

Challenge

These challenges extend the work we’ve done in the pracs and are a good test of your understanding.

1. Add class variables and methods to Dog, Cat and Bird to represent how they move (“moves”), and to getMoves(). These will be class variables as they will apply to all objects of that class.
2. Extend banking.py to have three bank accounts in a list. Modify your code to ask for the account for each transaction
3. Modify the program from challenge (2) to use a dictionary instead of a list