**SI 507**

**Homework 4:** Classes and Inheritance

## **Homework Objective:**

Demonstrate the ability to:

* Understand how Python classes contain
  + data representations (attributes, or instance variables)
  + ways to interact with class instances (methods)
* Become familiar with defining classes with and without inheritance
* Implement and use classes and inheritance
* Understand how and why to implement special class methods \_\_init\_\_ and \_\_str\_\_

## **Supporting Material:**

File: [hw4.py](https://www.dropbox.com/s/wi6qestw1wzooyf/hw4.py?dl=0)

Reading: [chapters on classes and inheritance](https://www.programsinformationpeople.org/runestone/static/publicPIP/index.html)

### **Part 1: Classes (10pts)**

#### **Task A: Exploration of a class (2pts)**

For Task A, we are going to explore a class, called *Explore\_pet*, which is already implemented. Please look through the class in the python file.

The class represents a generic pet that can get hungry and/or bored. Depending on the pet’s state, it will print different statements about itself. The pet becomes “bored” when its boredom value exceeds its boredom\_threshold, and it becomes “hungry” when its hunger value exceeds its hunger\_threshold.

You will see that an instance, *coco*, has been created. Alter the instance’s hunger and/or boredom value so that if we call print(*coco*), it would say “I’m Coco. I feel bored. You can teach me new words.” Now create another Pet called *brian* (with the name “Brian”) and set the values so that print(brian) will output “I’m Brian. I feel hungry. Feed me.”

#### **Task B: Implementation of Pet class (4pts)**

For the task B, we are going to implement *Pet* class so that we could use it for any kind of pet: dog, cat, and others. In the python file, you will find *Pet* class, which you will need to expand. After you finish that, we will use the class and interact with it on the task C. The *Pet* class should contain:

1. An additional instance variable
   1. words: a list of words that the pet knows
      1. Initially, this should be [“hello”].
2. Additional methods [remember to include the self parameter]
   1. clock\_tick( ): a method called when time passes
      1. An additional unit of 2 is added to the current hunger
      2. An additional unit of 2 is added to the current boredom
   2. say( )
      1. It should print “I know how to say: ” first.
      2. Then, it should print all the words the instance could say - one word per line
   3. teach(word)
      1. learns a new word
      2. boredom\_decrement is added to the current boredom; this becomes new boredom. If it goes to negative (boredom\_decrement is larger than the current boredom), then the current boredom becomes simply zero.
   4. feed( )
      1. hunger\_decrement is added to the current hunger; this becomes new hunger. If it goes to negative (hunger\_decrement is larger than the current hunger), then the current hunger becomes simply zero.

#### **Task C: Utilization of Pet class (4pts)**

For the task C, we are going to utilize the *Pet* class that we created in task B.

First, add the method hi( ) to *Pet*. When hi( ) is called, the *Pet* should say() (i.e., print) a random word from the list of words that it knows.

Second, complete the implementation of the teaching\_session( ) function. teaching\_session( ) takes two parameters: a Pet (in the parameter my\_pet) and a list of words (in the parameter new\_words). Within the function following things happen:

* my\_pet learns the new\_words
* For **each time** my\_pet learns a word from the new\_words,
  + my\_pet says a random word he/she knows (hint: use hi( ))
  + my\_pet prints out her current status (e.g., “I’m Fido and I feel happy” or “I’m Furball and I feel hungry. Feed me”)
  + if my\_pet is hungry, he/she is fed
  + The clock ticks once for my\_pet

Third, create a Pet (you choose the name!) and teach it ['I am sleepy', 'You are the best','I love you, too'] through the teaching\_session( ).

### **Part 2: Inheritance - subclasses (10pts)**

#### **Task A: Dog and Cat: subclasses of Pet (6pts)**

For the task A, we are going to implement the *Dog* and *Cat* subclasses.

* In the *Dog* class, dogs express their moods a bit differently. They always say "arrrf!" at the end. For example, instead of saying “I’m Coco. I feel happy.”, dogs express “I'm Coco, arrrf! I feel happy, arrrf!”; you would delete the period but add ", arrrf!" at the end of each sentence.
* In the *Cat* class, cats will have an additional instance variable, *meow\_count*. That is, Cat will take two parameters: *name* and *meow\_count*. We are also going to redefine the hi() method for cats. As opposed to superclass’s hi() method, cats will print the randomly chosen word for the number of *meow\_count* times. For example, if *meow\_count* was 3 and one random word was “hello”, it should print “hellohellohello”.

#### **Task B: Poodle - a subclass of Dog (4pts)**

First, we will implement a subclass of *Dog* class: *Poodle*.

* *Poodle*
  + Has an additional method called *dance*, which returns "Dancing in circles like poodles do!"
  + When the say method is executed, it will print the dance method first and then the say method from the superclass.

Second, we are going to create two instances and see if we get what is expected

1. Create a poodle (you choose the name!) and call the say( )method
2. Create a cat (you choose the name and meow\_count below 10) and call hi( )method

### **Extra credit 1 (2pts)**

For the first extra credit, you are going to implement a pet game. You need to create a new python file and save it as pet\_game.py. Running the file -- python3 pet\_game.py -- should automatically start the game. The pet game is a modified version of the textbook exercise and we encourage you to take a look at [the exercise code](https://www.programsinformationpeople.org/runestone/static/publicPIP/Inheritance/TamagotchiRevisited.html) and understand it. You should simply copy and paste the code from line 107 (def whichone…) to the end. You can ignore any class that you did not implement for the homework. Your pet game should follow additional rules along with the logic from the exercise code.

Additional rules

1. A new instance variable: self.age
   1. A pet will initially become an age of 0 and live up to 18 integer units (or years)
   2. *Dog* - increment of 2 for every clock\_tick
   3. *Cat* - increment of 3 for every clock\_tick
   4. Once either a cat or a dog gets older than 18 years, they will leave the world.
   5. Any method that needs to be overridden should preserve everything from the superclass.
2. A game ends when there is no pet except the beginning of the game where a player does not have any pets yet.
3. When the game ends, it should provide a player two options to make: play it again or quit.

### **Extra credit 2 (2pts)**

For the second extra credit, you are going to build *Library* class and subclasses and create Library simulation. Your library could have different types of media: reserve books, regular books, CDs, Movies, etc. Each media type should have different checkout policies; reserve books can be checked out for 1 integer unit (or day), regular books for 15 days, CDs for 7 days, Movies for 2 days.

First, you need to create a new python file and save it as sim\_library.py and running the file -- python3 sim\_library.py -- should automatically start the simulation.

you should do the followings:

1. Create a class and at least three subclasses representing different media types
   1. Possible attributes and methods to consider
      1. self.max\_days
      2. self.days\_passed or self.days\_remaining
      3. clock\_tick( ): the clock ticks, i.e. advances one day
      4. checkout( ): check an item out
      5. item\_return( ): return an item
      6. advance( ):move ahead the number of days
      7. overdue( ):check if it is overdue
2. When the library simulation begins,
   1. Catalog should contain at least 6 items - two instances from each subclass

**Sample run**

What would you like to do?

catalog: show the catalog

checkout <item\_number>

return <item\_number>

advance <num\_days>

account: see account status, including checked out items and overdue items

**What to turn in**:

All codes must be executable. Any code that does not run in Python3 will be given a score of 0. You can receive partial credit for working programs.

Submit following python module(s):

* hw4.py
* pet\_game.py (optional)
* sim\_library.py (optional)