ASSIGNMENT 3

Contents

[Contents 1](#_Toc471577617)

[Purpose 2](#_Toc471577618)

[Assignment Background 2](#_Toc471577619)

[Assignment Statement 2](#_Toc471577620)

[Requirements: 2](#_Toc471577621)

[What to Deliver 5](#_Toc471577622)

[Notes 5](#_Toc471577623)

[Grading 6](#_Toc471577624)

# Purpose

In this assignment, you will be demonstrating your understanding and use of python classes and inheritance and polymorphism, as covered in module 3.

# Assignment Background

This assignment makes use of inheritance to create a set of related classes. At the end, you’ll have a more complex structure of parent-child classes than you saw in the module’s examples.

# Assignment Statement

* You’ll create the following classes using inheritance:
  + A base class called Pet
  + A mix-in class called Jumper
  + A Dog class and a Cat class that each inherit from Pet and jumper
  + Two classes that inherit from Dog: BigDog and SmallDog
  + One classes that inherit from Cat: HouseCat
* The general skeleton of the Pet, Dog, and BigDog classes will be given to you, but you will have to identify the required inheritance for a class. You will be responsible for writing the functionality of those classes.
* A description of each class’s functionality, as well as the data to use, is in the requirements below.
* You will be required to write the code to exercise your classes per the requirements.

You are required to supply *Last\_Name\_First\_Name\_Assignment3.py.*

# Requirements:

Use the provided template to complete the following. The details are in the skeleton file.

1. The Pet class:
   1. Assign values to two variables: kind and color
   2. Implement the constructor to initialize the pet’s name
   3. Implement do\_tricks method per skeleton
2. The Jumper class
   1. Implement the jump method per the skeleton
3. The Dog class
   1. Decide which classes to inherit from and implement inheritance
   2. Change the kind to canine
   3. Implement \_\_str\_\_ per skeleton
   4. Implement \_\_call\_\_ per skeleton
4. The BigDog class
   1. Change the color to tan
   2. Implement \_\_str\_\_ per skeleton
   3. Implement speak per skeleton
5. The SmallDog class
   1. Change the color to brindle
   2. Implement \_\_str\_\_ per skeleton
   3. Implement speak per skeleton
6. The Cat class
   1. Change the kind to feline
   2. Implement \_\_str\_\_ per skeleton
   3. Implement speak per skeleton
   4. Implement climb per skeleton
7. The HouseCat class
   1. Change the color to white
   2. Implement \_\_str\_\_ per skeleton
   3. Implement speak per skeleton
8. Run the code according to the following:
   1. Instantiate each class(except jumper)
   2. Create a list of the instantiated objects
   3. Loop through the objects
   4. For each object:
      1. Print \_\_str\_\_
      2. print the kind of pet
      3. Print the Color of the pet
      4. Have the pet do tricks
      5. if applicable, print rollover action and the owners name
      6. If applicable, have the pet climb
      7. To separate each pet print underscores

Sample Output

Your output should look something like this

<\_\_main\_\_.Pet object at 0x105faf390>

animal

brown

Taz is doing tricks

----------------------------------------

I am a cat named Lion

None

feline

brown

Lion is doing tricks

Lion says Meow!!!

Lion is jumping

Lion is climbing the curtains again

----------------------------------------

I am a dog named Roo

None

canine

brown

Roo is doing tricks

Roo is jumping

Roo is rolling over

My owner is George

----------------------------------------

Noah is a large, muscular dog

canine

tan

Noah is doing tricks

Noah says Woof!!!

Noah is jumping

Noah is rolling over

My owner is George

----------------------------------------

Lucky is a tiny, cute dog

canine

brindle

Lucky is doing tricks

Lucky says Yip!

Lucky is jumping

Lucky is rolling over

My owner is George

----------------------------------------

Zebra is a cat with fluffy, white fur

feline

white

Zebra is doing tricks

Zebra says Purr

Zebra is jumping

Zebra is climbing the curtains again

----------------------------------------

Code/Comment Format

Good code includes well named variables that are consistent from the beginning to the end of the program. Naming of objects should be self-explanatory. For instance, iterator\_for\_noun\_list is much better than i.

Every program consists of a sequence of paragraphs, each of which has objectives, and which builds on the previous paragraphs. We are mostly interested in objectives that are valid at the end of the program so we can verify the program's design. The following is a preferred form for such paragraph headings. The # sign is adequate when the comment is a single line.

#This is an in-line comment – used to document the code for you, or anyone else, that intends

#To extend the code

In-line comments are helpful when one has to go back to the code 6 months later to make changes.

For doc strings, python allows the use of triple quotes. The triple quotes can be either single or double quotes. A doc sting is generally used as user documentation. It does not need to include details of the implementation of the program, but instead it provides documentation as how to use the API for the program (input, output etc.)

For example:

“””

This is an example of a doc string

It allows multiple lines within the string.

“””

‘’’

This is an example of a doc string

It allows multiple lines within the string.

‘’’

This becomes significant when using functions, classes etc. as the triple quotes help to self-document the parameters and return values of the function.

# What to Deliver

Supply

The *Last Name\_First Name\_Assignment3.py* file.

# Notes

* Assignments can be submitted once. If extenuating circumstances exist, contact your facilitator.
* Note the statement in the syllabus on timeliness of submissions (the gist being that all assignments must observe the deadlines).
* Start by identifying and ordering the objectives.
* There are no testing requirements for this assignment. However, it would be prudent to make sure your program does not crash and all input validation is performed correctly.

# Grading

Pet class:

1. Variable assignment – 2 points
2. Implement constructor – 2 points
3. Implement do\_tricks
   1. Print statement – 2 points
   2. Call speak method – 2 points
   3. Call jump method – 2 points

Jumper Class

1. Implement jump – 2 points

Dog Class

1. Add inheritance – 5 points
2. Change kind variable – 2 points
3. Implement \_\_str\_\_ - 2 points
4. Implement \_\_call\_\_ - 5 points

BigDog Class

1. Add inheritance – 2 point
2. Change color to tan – 2 points
3. Implement \_\_str\_\_ - 2 points
4. Implement speak – 2 points

SmallDog class

1. Add inheritance – 2 point
2. Change color to brindle– 2 points
3. Implement \_\_str\_\_ - 2 points
4. Implement speak – 2 points

Cat class

1. Add inheritance – 2 points
2. Change kind to feline – 2 points
3. Implement \_\_str\_\_ - 2 points
4. Implement speak – 2 points
5. Implement climb – 2 points

HouseCat class

1. Add inheritance – 2 point
2. Change color to white– 2 points
3. Implement \_\_str\_\_ - 2 points
4. Implement speak – 2 points

Use your code

1. Instantiate each class(except jumper) – 4 points
2. Create a list of the instantiated objects – 1 points
3. Loop through the objects – 2 points
4. Print \_\_str\_\_ - 4 points
5. print the kind of pet – 2 points
6. Print the Color of the pet – 2 points
7. Have the pet do tricks – 8 points
8. if applicable, print rollover action and the owners name – 8 points
9. If applicable, have the pet climb – 8 points
10. To separate each pet print underscores – 1 points