ASSIGNMENT 3

Contents

[Purpose 1](#__RefHeading___Toc338_70214081)

[Assignment Background 1](#__RefHeading___Toc340_70214081)

[Assignment Statement 1](#__RefHeading___Toc342_70214081)

[Requirements: 2](#__RefHeading___Toc344_70214081)

[What to Deliver 6](#__RefHeading___Toc346_70214081)

[Notes 6](#__RefHeading___Toc348_70214081)

[Grading 6](#__RefHeading___Toc350_70214081)

# 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:
  + An abstract 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 class that inherits from Cat: HouseCat
* 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.

# Requirements:

1. The Pet class has:
   1. An initializer (\_\_init\_\_)
      1. The initializer requires the following parameters: name and color.
      2. Assigns the parameters to instance members with the same names (*name* and *color*)
   2. A do\_tricks instance method
      1. It takes no parameters
      2. It prints “{name of the pet} is doing tricks.”
      3. It invokes the instance method speak.
   3. A speak instance method
      1. It takes no parameters
      2. It raises NotImplementedError
2. The Jumper class has
   1. A jump instance method
      1. It takes the following parameter: name
      2. It prints “{the name parameter} is jumping.”
3. The Dog class
   1. Inherits from Jumper and Pet

*HINT: a mixin class must be inherited before a parent*

* 1. Has an initializer
     1. which requires the same parameters as the parent class Pet as well as one additional parameter: owner
     2. which invokes the super initializer with the appropriate parameters
     3. which assigns the parameter owner to a instance member with the same name (*owner*)
     4. which assigns the instance member *kind* to the string “canine”
  2. Has a \_\_str\_\_ instance method
     1. It takes no parameters
     2. It returns the string “I am a dog named {name of the pet} and my owner is {name of the owner}”
  3. Has a \_\_call\_\_ instance method
     1. It takes a single parameter: action
     2. If the action is “rollover”: returns the string “{name of the pet} is rolling over.”
     3. If the action is “tracking”: returns the string “{name of the pet} is tracking.”
  4. Has do\_tricks instance method
     1. It takes no parameters
     2. It invokes the super do\_tricks instance method
     3. It invokes jump instance method with the pet’s name as its argument.
  5. Has a speak instance method
     1. It takes no parameter
     2. It prints “{name of the pet} says Woof Woof!!!”

1. The BigDog class
   1. Inherits from Dog
   2. Has a \_\_str\_\_ instance method
      1. It takes no parameters
      2. It returns the string “'{name of the pet} is a large, {color of the pet} muscular dog'”
   3. Has a speak instance method
      1. It takes no parameter
      2. It prints “{name of the pet} says Woof!!!”
2. The SmallDog class
   1. Inherits from Dog
   2. Has a \_\_str\_\_ instance method
      1. It takes no parameters
      2. It returns the string “'{name of the pet} is a tiny, cute dog'”
   3. Has a speak instance method
      1. It takes no parameter
      2. It prints “{name of the pet} says Yip!!!”
3. The Cat class
   1. Inherits from Jumper and Pet

*HINT: a mixin class must be inherited before a parent*

* 1. Has an initializer
     1. which requires the same parameters as the parent class Pet
     2. which invokes the super initializer with the appropriate parameters
     3. which assigns the instance member *kind* to the string “feline”
  2. Has a \_\_str\_\_ instance method
     1. It takes no parameters
     2. It returns the string “I am a cat named {name of the pet}, I have no owner!”
  3. Has do\_tricks instance method
     1. It takes no parameters
     2. It invokes the super do\_tricks instance method
     3. It invokes jump instance method with the pet’s name as its argument.
  4. Has a speak instance method
     1. It takes no parameter
     2. It prints “{name of the pet} says Meow!!!”
  5. Has a climb instance method
     1. It takes no parameter
     2. It prints “{name of the pet} is climbing the curtains again.”

1. The HouseCat class
   1. Inherits from Cat
   2. Has a \_\_str\_\_ instance method
      1. It takes no parameters
      2. It returns the string “'{name of the pet} is a cat with fluffy, white fur'”
   3. Has a speak instance method
      1. It takes no parameter
      2. It prints “{name of the pet} says Purr!!!”
2. Run the code according to the following:
   1. Instantiate each class(except Jumper and Pet)
      1. You are required to supply owner, name and color
   2. Create a list of the instantiated objects (1 Dog, 1 Cat, 1 BigDog, 1 SmallDog, 1 HouseCat)
   3. For each object in the list:
      1. Print the string representation of the object
      2. Print the name, and kind of pet
      3. Print the name, and color of the pet
      4. If applicable, have the pet do tricks
      5. if applicable, print rollover action and tracking action

*Hint: verify that the instance supports the operation*

* + 1. If applicable, have the pet climb

*Hint: verify that the instance supports the operation*

* + 1. Separate each pet’s print with 42 underscores

Sample Output

Your output should look something like this. This code may not be a complete representation. It is not intended to reverse engineer. It is intended to guide you as to whether you are on the right track.

I am a cat named Lion, I have no owner

Lion's kind is feline.

Lion's color is Tan.

Lion is doing tricks

Lion says Meow!!!

Lion is jumping

Lion is climbing the curtains again

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

I am a dog named Roo and my owner is jane

Roo's kind is canine.

Roo's color is brindle.

Roo is doing tricks

Roo says Woof Woof!!!

Roo is jumping

Roo is rolling over

Roo is tracking.

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

Noah is a large, Albino muscular dog

Noah's kind is canine.

Noah's color is Albino.

Noah is doing tricks

Noah says Woof!!!

Noah is jumping

Noah is rolling over

Noah is tracking.

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

Lucky is a tiny, cute dog

Lucky's kind is canine.

Lucky's color is black.

Lucky is doing tricks

Lucky says Yip!

Lucky is jumping

Lucky is rolling over

Lucky is tracking.

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

Zebra is a cat with fluffy, white fur

Zebra's kind is feline.

Zebra's color is Striped.

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

You are required to supply a single file - *Last\_Name\_First\_Name\_Assignment3.py.*

# 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. Implement constructor – 2 points

2. Variable assignment – 2 points

3. Implement do\_tricks

a. Print statement – 2 points

b. Call speak method – 2 points

3. implement speak – 2 points

Jumper Class

4. Implement jump – 2 points

Dog Class

5. Add inheritance – 1 points

6. Set kind variable – 2 points

7. Implement \_\_str\_\_ – 2 points

8. Implement \_\_call\_\_ – 4 points

9. Implement do\_tricks – 4 points

10. Implement speak – 2 points

BigDog Class

9. Add inheritance – 2 point

11. Implement \_\_str\_\_ – 2 points

12. Implement speak – 2 points

SmallDog class

13. Add inheritance – 2 point

15. Implement \_\_str\_\_ – 2 points

16. Implement speak – 2 points

Cat class – 14 pts

17. Add inheritance – 2 points

18. Set kind to feline – 2 points

19. Implement \_\_str\_\_ – 2 points

21. Implement do\_tricks – 4 points

20. Implement speak – 2 points

21. Implement climb – 2 points

HouseCat class

22. Add inheritance – 2 point

24. Implement \_\_str\_\_ – 2 points

25. Implement speak – 2 points

Use your code

26. Instantiate each class (except Jumper and Pet) – 4 points

27. Create a list of the instantiated objects – 1 points

28. Loop through the objects – 2 points

29. Print \_\_str\_\_ – 4 points

30. print the name and kind of pet – 2 points

30. print the name and color of pet – 2 points

31. Have the pet do tricks – 8 points

32. if applicable, print rollover action and tracking action– 8 points

33. If applicable, have the pet climb – 8 points

34. To separate each pet print underscores – 2 points