Inheritance - Lab

Introduction

In this lab, you'll use what you've learned about inheritance to model a zoo using superclasses, subclasses, and maybe even an abstract superclass!

Objectives

In this lab you will:

- Create a domain model using OOP
- Use inheritance to write nonredundant code

Modeling a Zoo

Consider the following scenario: You've been hired by a zookeeper to build a program that keeps track of all the animals in the zoo. This is a great opportunity to make use of inheritance and object-oriented programming!

Creating an Abstract Superclass

Start by creating an abstract superclass, `Animal()`. When your program is complete, all subclasses of `Animal()` will have the following attributes:

- `name`, which is a string set at instantation time
- `size`, which can be `'small'`, `'medium'`, `'large'`, or `'enormous'`
- `weight`, which is an integer set at instantiation time
- `species`, a string that tells us the species of the animal
- `food_type`, which can be `'herbivore'`, `'carnivore'`, or `'omnivore'`
- `nocturnal`, a boolean value that is `True` if the animal sleeps during the day, otherwise `False`

They'll also have the following behaviors:

- `sleep`, which prints a string saying if the animal sleeps during day or night
- `eat`, which takes in the string `'plants'` or `'meat'`, and returns `'{animal name} the {animal species} thinks {food} is yummy!'` or `'I don't eat this!'` based on the animal's `food_type` attribute

In the cell below, create an abstract superclass that meets these specifications.

NOTE: For some attributes in an abstract superclass such as `size`, the initial value doesn't matter -- just make sure that you remember to override it in each of the subclasses!

```
class Animal(object):

    def __init__(self, name, weight):
        self.name = name
        self.weight = weight
        self.species = None
        self.size = None
        self.food_type = None
        self.nocturnal = False

    def sleep(self):
```

```
            if self.nocturnal:
                print("{} sleeps during the day!".format(self.name))
            else:
                print("{} sleeps during the night!".format(self.name))

        def eat(self, food):
            if self.food_type == 'omnivore':
                print("{} the {} thinks {} is Yummy!".format(self.name, self.species, fo
            elif (food == 'meat' and self.food_type == "carnivore") or (food == 'plants'
                print("{} the {} thinks {} is Yummy!".format(self.name, self.species, fo
            else:
                print("I don't eat this!")
```

Great! Now that you have our abstract superclass, you can begin building out the specific animal classes.

In the cell below, complete the `Elephant()` class. This class should:

- subclass `Animal`
- have a species of `'elephant'`
- have a size of `'enormous'`
- have a food type of `'herbivore'`
- set nocturnal to `False`

Hint: Remember to make use of `.super()` during initialization, and be sure to pass in the values it expects at instantiation time!

```
class Elephant(Animal):

    def __init__(self, name, weight):
        super().__init__(name, weight)
        self.size = 'enormous'
        self.species = 'elephant'
        self.food_type = 'herbivore'
        self.nocturnal = False
```

Great! Now, in the cell below, create a `Tiger()` class. This class should:

- subclass `Animal`
- have a species of `'tiger'`
- have a size of `'large'`
- have a food type of `'carnivore'`

- set nocturnal to `True`

```
class Tiger(Animal):

    def __init__(self, name, weight):
        super().__init__(name, weight)
        self.size = 'large'
        self.species = 'tiger'
        self.food_type = 'carnivore'
        self.nocturnal = True
```

Great! Two more classes to go. In the cell below, create a `Raccoon()` class. This class should:

- subclass `Animal`
- have a species of `raccoon`
- have a size of `'small'`
- have a food type of `'omnivore'`
- set nocturnal to `True`

```
class Raccoon(Animal):

    def __init__(self, name, weight):
        super().__init__(name, weight)
        self.size = 'small'
        self.species = 'raccoon'
        self.food_type = 'omnivore'
        self.nocturnal = True
```

Finally, create a `Gorilla()` class. This class should:

- subclass `Animal`
- have a species of `gorilla`
- have a size of `'large'`
- have a food type of `'herbivore'`
- set nocturnal to `False`

```
class Gorilla(Animal):

    def __init__(self, name, weight):
        super().__init__(name, weight)
        self.size = 'large'
```

```
        self.species = 'gorilla'
        self.food_type = 'herbivore'
        self.nocturnal = False
```

Using Our Objects

Now it's time to populate the zoo! To ease the creation of animal instances, create a function `add_animal_to_zoo()`.

This function should take in the following parameters:

- `zoo`, an array representing the current state of the zoo
- `animal_type`, a string. Can be `'Gorilla'`, `'Raccoon'`, `'Tiger'`, or `'Elephant'`
- `name`, the name of the animal being created
- `weight`, the weight of the animal being created

The function should then:

- use `animal_type` to determine which object to create
- Create an instance of that animal, passing in the `name` and `weight`
- Append the newly created animal to `zoo`
- Return `zoo`

```
def add_animal_to_zoo(zoo, animal_type, name, weight):
    animal = None
    if animal_type == 'Gorilla':
        animal = Gorilla(name, weight)
    elif animal_type == 'Raccoon':
        animal = Raccoon(name, weight)
    elif animal_type == 'Tiger':
        animal = Tiger(name, weight)
    else:
        animal = Elephant(name, weight)

    zoo.append(animal)

    return zoo
```

Great! Now, add some animals to your zoo.

Create the following animals and add them to your zoo. The names and weights are up to you.

- 2 Elephants
- 2 Raccons
- 1 Gorilla
- 3 Tigers

```
to_create = ['Elephant', 'Elephant', 'Raccoon', 'Raccoon', 'Gorilla', 'Tiger', 'Tige

zoo = []

for i in to_create:
    zoo = add_animal_to_zoo(zoo, i, 'name', 100)

zoo
```

```
[<__main__.Elephant at 0x24360cc2a20>,
 <__main__.Elephant at 0x24360cc2470>,
 <__main__.Raccoon at 0x24360cc2208>,
 <__main__.Raccoon at 0x24360cc21d0>,
 <__main__.Gorilla at 0x24360cc2278>,
 <__main__.Tiger at 0x24360cc2710>,
 <__main__.Tiger at 0x24360cc2780>,
 <__main__.Tiger at 0x24360cc2eb8>]
```

Great! Now that you have a populated zoo, you can do what the zookeeper hired you to do -- write a program that feeds the correct animals the right food at the right times!

To do this, write a function called `feed_animals()`. This function should take in two arguments:

- `zoo`, the zoo array containing all the animals
- `time`, which can be `'Day'` or `'Night'`. This should default to day if nothing is entered for `time`

This function should:

- Feed only the non-nocturnal animals if `time='Day'`, or only the nocturnal animals if `time='Night'`
- Check the food type of each animal before feeding. If the animal is a carnivore, feed it `'meat'`; otherwise, feed it `'plants'`. Feed the animals by using their `.eat()` method

```
def feed_animals(zoo, time='Day'):
    for animal in zoo:
        if time == 'Day':
            # CASE: Daytime feeding -- Only feed the animals that aren't nocturnal
            if animal.nocturnal == False:
                # If the animal is a carnivore, feed it "meat".  Otherwise, feed it
                if animal.food_type == 'carnivore':
                    animal.eat('meat')
                else:
                    animal.eat('plants')
        else:
            # CASE: Night-time feeding -- feed only the nocturnal animals!
            if animal.nocturnal == True:
                if animal.food_type == 'carnivore':
                    animal.eat('meat')
                else:
                    animal.eat('plants')
```

Now, test out your program. Call the function for a daytime feeding below.

```
feed_animals(zoo)
```

```
name the elephant thinks plants is Yummy!
name the elephant thinks plants is Yummy!
name the gorilla thinks plants is Yummy!
```

If the elephants and gorrillas were fed then things should be good!

In the cell below, call `feed_animals()` again, but this time set `time='Night'`

```
feed_animals(zoo, 'Night')
```

```
name the raccoon thinks plants is Yummy!
name the raccoon thinks plants is Yummy!
name the tiger thinks meat is Yummy!
name the tiger thinks meat is Yummy!
name the tiger thinks meat is Yummy!
```

That's it! You've used OOP and inheritance to build a working program to help the zookeeper feed his animals with right food at the correct times!

Summary

In this lab, you modeled a zoo and learned how to use inheritance to write nonredundant code, used subclasses and superclasses, and create a domain model using OOP.

Releases

No releases published

Packages

No packages published

Contributors 5

Languages

- **Jupyter Notebook** 100.0%