22.1. Introduction: Class Inheritance

Classes can "inherit" methods and class variables from other classes. We'll see the mechanics of how this works in subsequent sections. First, however, let's motivate why this might be valuable. It turns out that inheritance doesn't let you do anything that you couldn't do without it, but it makes some things a lot more elegant. You will also find it's useful when someone else has defined a class in a module or library, and you just want to override a few things without having to reimplement everything they've done.

Consider our Tamagotchi game. Suppose we wanted to make some different kinds of pets that have the same structure as other pets, but have some different attributes or behave a little differently. For example, suppose that dog pets should show their emotional state a little differently than cats or act differently when they are hungry or when they are asked to fetch something.

You could implement this by making an instance variable for the pet type and dispatching on that instance variable in various methods.

```
from random import randrange

class Pet():
    boredom_decrement = 4
    hunger_decrement = 6
    boredom_threshold = 5
    hunger_threshold = 10
    sounds = ['Mrrp']
    def __init__(self, name = "Kitty", pet_type="dog"):
        self.name = name
        self.hunger = randrange(self.hunger_threshold)
        self.boredom = randrange(self.boredom_threshold)
        self.sounds = self.sounds[:]  # copy the class attribute, so that when we make cha
nges to it, we won't affect the other Pets in the class
        self.pet_type = pet_type

    def clock_tick(self):
        self.boredom += 1
        self.hunger += 1

    def mood(self):
        if self.hunger <= self.hunger_threshold and self.boredom <= self.boredom_threshol
d:
            if self.pet_type == "dog": # if the pet is a dog, it will express its mood in
different ways from a cat or any other type of animal
                return "happy"
            elif self.pet_type == "cat":
                return "happy, probably"
            else:
                return "HAPPY"
        elif self.hunger > self.hunger_threshold:
            if self.pet_type == "dog": # same for hunger -- dogs and cats will express the
ir hunger a little bit differently in this version of the class definition
                return "hungry, arf"
            elif self.pet_type == "cat":
                return "hungry, meeeeow"
            else:
                return "hungry"
        else:
            return "bored"

    def __str__(self):
        state = "     I'm " + self.name + ". "
        state += " I feel " + self.mood() + ". "
        return state

    def hi(self):
        print(self.sounds[randrange(len(self.sounds))])
        self.reduce_boredom()

    def teach(self, word):
        self.sounds.append(word)
        self.reduce_boredom()

    def feed(self):
        self.reduce_hunger()

    def reduce_hunger(self):
        self.hunger = max(0, self.hunger - self.hunger_decrement)

    def reduce_boredom(self):
        self.boredom = max(0, self.boredom - self.boredom_decrement)
```

That code is exactly the same as the code defining the `Pet` class that you saw in the Tamagotchi section, except that we've added a few things.

- A new input to the constructor – the `pet_type` input parameter, which defaults to `"dog"`, and the `self.pet_type` instance variable.
- if..elif..else in the `self.mood()` method, such that different types of pets (a dog, a cat, or any other type of animal) express their moods and their hunger in slightly different ways.

But that's not an elegant way to do it. It obscures the parts of being a pet that are common to all pets and it buries the unique stuff about being a dog or a cat in the middle of the mood method. What if you also wanted a dog to reduce boredom at a different rate than a cat, and you wanted a bird pet to be different still? Here, we've only implemented **dogs**, **cats**, and **other** – but you can imagine the possibilities.

If there were lots of different types of pets, those methods would start to have long and complex **if..elif..elif** code clauses, which can be confusing. And you'd need that in every method where the behavior was different for different types of pets. Class inheritance will give us a more elegant way to do it.

You have attempted 1 of 1 activities on this page

22. Inheritance">
2. Inheritance">
22.2. Inheriting Variables and Methods">
Completed. Well Done!
22.2. Inheriting Variables and Methods">Next Section - 22.2.