Exercise 42: Is-A, Has-A, Objects, and Classes

An important concept that you have to understand is the difference between a `class` and an `object`. The problem is, there is no real "difference" between a `class` and an `object`. They are actually the same thing at different points in time. I will demonstrate by a Zen koan:

```
What is the difference between a Fish and a Salmon?
```

Did that question sort of confuse you? Really sit down and think about it for a minute. I mean, a Fish and a Salmon are different but, wait, they are the same thing, right? A Salmon is a *kind* of Fish, so I mean it's not different. But at the same time, because a Salmon is a particular *type* of Fish and so it's actually different from all other Fish. That's what makes it a Salmon and not a Halibut. So a Salmon and a Fish are the same but different. Weird.

This question is confusing because most people do not think about real things this way, but they intuitively understand them. You do not need to think about the difference between a Fish and a Salmon because you *know* how they are related. You know a Salmon is a *kind* of Fish and that there are other kinds of Fish without having to understand that.

Let's take it one step further, let's say you have a bucket full of three Salmon and because you are a nice person, you have decided to name them Frank, Joe, and Mary. Now, think about this question:

```
What is the difference between Mary and a Salmon?
```

Again this is a weird question, but it's a bit easier than the Fish vs. Salmon question. You know that Mary is a Salmon, and so she's not really different. She's just a specific "instance" of a Salmon. Joe and Frank are also instances of Salmon. What do I mean when I say instance? I mean they were created from some other Salmon and now represent a real thing that has Salmon-like attributes.

Now for the mind-bending idea: Fish is a `class`, and Salmon is a `class`, and Mary is an `object`. Think about that for a second. Alright let's break it down real slow and see if you get it.

A Fish is a `class`, meaning it's not a *real* thing, but rather a word we attach to instances of things with similar attributes. Got fins? Got gills? Lives in water? Alright it's probably a Fish.

Someone with a Ph.D. then comes along and says, "No, my young friend, *this* Fish is actually *Salmo salar*, affectionately known as a Salmon." This professor has just clarified the Fish further; and made a new `class` called "Salmon" that has more specific attributes. Longer nose, reddish flesh, big, lives in the ocean or fresh water, tasty? Ok, probably a Salmon.

Finally, a cook comes along and tells the Ph.D., "No, you see this Salmon right here, I'll call her Mary and I'm going to make a tasty fillet out of her with a nice sauce." Now you have this *instance* of a Salmon (which also is an instance of a Fish) named Mary turned into something real that is filling your belly. It has become an `object`.

There you have it: Mary is a kind of Salmon that is a kind of Fish. `object` is a `class` is a `class`.

How This Looks in Code

This is a weird concept, but to be very honest you only have to worry

about it when you make new classes, and when you use a class. I will show you two tricks to help you figure out whether something is a `class` or `object`.

First, you need to learn two catch phrases "`is-a`" and "`has-a`." You use the phrase `is-a` when you talk about objects and classes being related to each other by a class relationship. You use `has-a` when you talk about objects and classes that are related only because they *reference* each other.

Now, go through this piece of code and replace each `##??` comment with a replacement comment that says whether the next line represents an `is-a` or a `has-a` relationship and what that relationship is. In the beginning of the code, I've laid out a few examples, so you just have to write the remaining ones.

Remember, `is-a` is the relationship between Fish and Salmon, while `has-a` is the relationship between Salmon and Gills.

```
## Animal is-a object (yes, sort of confusing) look at
the extra credit
class Animal(object):
    pass

## ??
class Dog(Animal):

    def __init__(self, name):
        ## ??
        self.name = name

## ??
class Cat(Animal):

    def __init__(self, name):
        ## ??
        self.name = name

## ??
class Person(object):

    def __init__(self, name):
```

```
24          ## ??
25          self.name = name
26
27          ## Person has-a pet of some kind
28          self.pet = None
29
30  ## ??
31  class Employee(Person):
32
33      def __init__(self, name, salary):
34          ## ?? hmm what is this strange magic?
35          super(Employee, self).__init__(name)
36          ## ??
37          self.salary = salary
38
39  ## ??
40  class Fish(object):
41      pass
42
43  ## ??
44  class Salmon(Fish):
45      pass
46
47  ## ??
48  class Halibut(Fish):
49      pass
50
51
52  ## rover is-a Dog
53  rover = Dog("Rover")
54
55  ## ??
56  satan = Cat("Satan")
57
58  ## ??
59  mary = Person("Mary")
60
61  ## ??
62  mary.pet = satan
63
64  ## ??
65  frank = Employee("Frank", 120000)
66
67  ## ??
68  frank.pet = rover
69
70  ## ??
71  flipper = Fish()
72
73  ## ??
74  crouse = Salmon()
75
76  ## ??
    harry = Halibut()
```

About class Name(object)

Remember how I was yelling at you to always use `class Name(object)` and I couldn't tell you why? Now I can tell you, because you just learned about the difference between a `class` and an `object`. I couldn't tell you until now because you would have just been confused and couldn't learn to use the technology.

What happened is Python's original rendition of `class` was broken in many serious ways. By the time they admitted the fault it was too late, and they had to support it. In order to fix the problem, they needed some "new class" style so that the "old classes" would keep working but you could use the new more correct version.

This is where "`class is-a object`" comes in. They decided that they would use the word "object," lowercased, to be the "class" that you inherit from to make a class. Confusing, right? A `class` inherits from the `class` named `object` to make a `class` but it's not an `object` really it's a `class`, but do not forget to inherit from `object`.

Exactly. The choice of one single word meant that I couldn't teach you about this until now. Now you can try to understand the concept of a class that is an `object` if you like.

However, I would suggest you do not. Just completely ignore the idea of old style vs. new style classes and assume that Python always requires (object) when you make a `class`. Save your brain power for something important.

Study Drills

1. Research why Python added this strange `object` class, and what that means.

2. Is it possible to use a `class` like it's an `object`?
3. Fill out the animals, fish, and people in this exercise with functions that make them do things. See what happens when functions are in a "base class" like Animal vs. in say Dog.
4. Find other people's code and work out all the `is-a` and `has-a` relationships.
5. Make some new relationships that are lists and dicts so you can also have "has-many" relationships.
6. Do you think there's a such thing as an "is-many" relationship? Read about "multiple inheritance," then avoid it if you can.

Common Student Questions

What are these `## ??` comments for?

Those are "fill-in-the-blank" comments that you are supposed to fill in with the right "`is-a`," "`has-a`" concepts. Re-read this exercise and look at the other comments to see what I mean.

What is the point of `self.pet = None`?

That makes sure that the `self.pet` attribute of that `class` is set to a default of `None`.

What does `super(Employee, self).__init__(name)` do?

That's how you can run the `__init__` method of a parent class reliably. Go search for "python super" and read the various advice on it being evil and good for you.

- All 52 videos, 1 per exercise, almost 2G of video.
- A PDF of the book.
- Email help from the author.
- See a list of everything you get before you buy.

When you buy the videos they will immediately show up **right here** without any hassles.

Already Paid? Reactivate Your Purchase Right Now!

Buying Is Easy

Buying is easy. Just fill out the form below and we'll get started.

Full Name

Email Address

Pay With Credit Card (by Stripe™)

Use your PayPal™ account.

Buy Learn Python The Hard Way, 3rd Edition