INFO5002: Intro to Python for Info Sys

OOP

PCC 167-172

Northeastern University

Slides created by: Zachary Doucet

The holy grail

“Everything is an object. Treat everything like an object.”

Source: Monty Python and the Holy Grail

Pizza Restaurant

- Let’s say I am opening a pizza restaurant off Granville and West Georgia. What classes would I create to model this restaurant?

Source: Arnold Gatilao

Source: Wikimedia

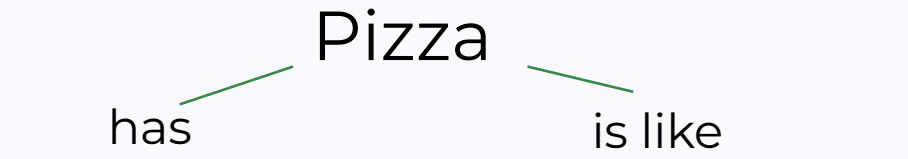
Class for each object

- Generally, every physical entity should have its own class. E.g. a table, a car, a television, a shirt, etc.
- Similarly, non-physical entities should have their own class. E.g. a lecture, an idea, a law, a game, etc.

Any possible situation can be modelled as a group of objects with class definitions.

How objects relate

- If an object has another, use an attribute.
- If an object is/is like another, use inheritance.

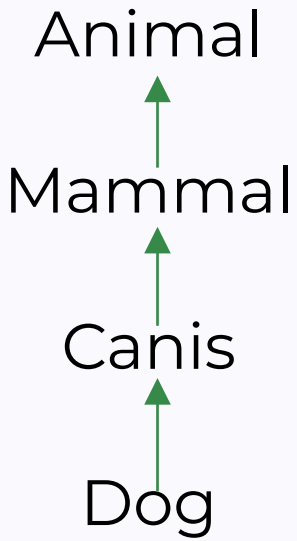


Source: Eva K.

Source: Deryck Chan

Inheritance

- Inheritance is a way to save code by copying all the **methods and attributes** from the parent to the child.



Inheriting

- You inherit a class by placing the name of the parent class in parentheses after the child's class name declaration.

```
class Animal:
    def __init__(self, name):
        self.name = name
```

```
animal = Animal("Timmy")
print(animal.name)
```

```
class Dog(Animal):
    def woof():
        print("Woof!")
```

```
dog = Dog("Timmy")
print(dog.name)
Dog.woof()
```

Not everything gets inherited

- Any methods that you define in the child's class that exist in the parent's class will not be inherited.

```
class Animal:
    def __init__(self, name):
        self.name = name
```

Only the method's name matters in determining whether to inherit or not.

```
class Dog(Animal):
    def __init__(self, name, breed):
        super().__init__(name)
        self.breed = breed

    def woof():
        print("Woof!")
```

Keep super call first statement.

Inheritance can make objects behave differently

- Two objects may have a common superclass with a common method which behaves differently (Polymorphism).

```
class Animal:
    def __init__(self, name):
        self.name = name

    def speak(self):
        print("Squeek")
```

```
class Dog(Animal):
    def speak(self):
        print("Woof!")
```

```
class Cat(Animal):
    def speak(self):
        print("Meow.")
```

```
x = Animal()
y = Dog()
z = Cat()
x.speak()
y.speak()
z.speak()
```