Introduction to Programming W15

Introduction to Object-Oriented Programming in Python II.

2022/07/15

1 Inheritance, overriding methods

A class can take the attributes and methods of another class. This process is called **inheritance**. The newly formed classes are called **child classes**, while the original classes where the child classes are derived from are the **parent classes**. Child classes can extend or even override parent class methods and attributes. In this example, the class `Cat` will be our parent class, and `Persian`, `Siamese`, and `Munchkin` are the child classes. To create a child class, you need to put the parent class name into the parentheses in the child class definition. Try the following code:

```
class Cat:
    animal_type = "carnivore"

    def __init__(self, name, color, hair, age):
        self.name = name
        self.color = color
        self.hair = hair
        self.age = age

    def say(self, sound):
        return "{} says {}".format(self.name, sound)

class Persian(Cat):
    pass

class Siamese(Cat):
    pass

class Munchkin(Cat):
    pass

luna = Persian("Luna", "white", "long", 2)
milo = Siamese("Milo", "gray", "short", 4)
oliver = Munchkin("Oliver", "black", "short", 1)

#instead of printing everything one-by-one, we create a list a loop through
cats = [luna, milo, oliver]
#remember string concatenation and conditionals
for cat in cats:
    print(cat.name, "is", cat.color+",", cat.hair, "haired", "and", cat.age, "years old.")
    if cat.name == "Luna":
        print(cat.say("meow"))
    else:
        print(cat.say("nyan"))
```

In the above example, Persian cats say "meow", and both Siamese and Munchkin say "nyan". We can *override* the `.say()` method in the child class definitions, and provide a default values.

```
class Cat:
    animal_type = "carnivore"

    def __init__(self, name, color, hair, age):
        self.name = name
        self.color = color
        self.hair = hair
        self.age = age

    def say(self, sound):
        return "{} says {}".format(self.name, sound)


class Persian(Cat):
    #only change the say method, other attributes and methods are the same
    def say(self, sound="meow"):
        return "{} says {}".format(self.name, sound)

class Siamese(Cat):
    def say(self, sound="nyan"):
        return "{} says {}".format(self.name, sound)
class Munchkin(Cat):
    def say(self, sound="nyan"):
        return "{} says {}".format(self.name, sound)

luna = Persian("Luna", "white", "long", 2)
milo = Siamese("Milo", "gray", "short", 4)
oliver = Munchkin("Oliver", "black", "short", 1)

print(luna.say())
print(milo.say())
print(oliver.say())
#we can still pass unique values
print(oliver.say("nyaaan"))
```

2 Adding new attributes, inheriting from the constructor

I we want to add new instance attributes to the child class, we need to used the `super()` function to inherit the attributes from the parent class, or the child's `.__init__()` constructor would override the parent's `.__init__()` constructor (which would be valid code as well). Try the following code and pay attention to the comments.

```
class Cat:
    """
    This is a class docstring. It describes the class,
    potentially the arguments/parameters and their types.
    """
    animal_type = "carnivore"

    def __init__(self, name, color, hair, age):
        self.name = name
        self.color = color
        self.hair = hair
        self.age = age
        #this is still an instance attribute, but cannot be passed as an argument:
        self.property = "cute"
```

```
    def describe(self):
        """
        This is a function docstring.
        """
        return "{} is {} years old".format(self.name, self.age)

    def say(self, sound):
        return "{} says {}".format(self.name, sound)


class Munchkin(Cat):
    #new class attribute for Munchkin, this will be the same for all
    #objects of the class:
    characteristic = "very short legs"

    #we would like to add a new instance attribute "weight"
    #also, make "short" for default hair
    def __init__(self, name, color, age, weight, hair="short"):
        super().__init__(name, color, hair, age) #inherit from parent class
        self.weight = weight #the new attribute, for the Munchkin class

    #overriding the say method from Cat:
    def say(self, sound="nyan"):
        return "{} says {}".format(self.name, sound)

    #completely new method for the Munchkin class, not available for Cat
    def cat_info(self):
        info_dict = {"name":self.name, "color":self.color,
                     "age":self.age, "hair":self.hair}
        return info_dict

    #new method for the Munchkin class, which uses the describe method from Cat
    def another_describe(self):
        return super().describe() + " and " + self.color

oliver = Munchkin("Oliver", "black", 1, 2.4) #didn't have to pass hair
print(oliver.name+"'s hair is", oliver.hair) #using the default value
#we can still change the default value:
loki = Munchkin("Loki", "black", 7, 4.2, "long")
print(loki.name+"'s hair is", loki.hair)
print(loki.describe()) #we kept the describe method from the parent class Cat
print(loki.another_describe()) #the new describe method only for Munchkin
#printing class attributes animal type and characteristic
print(loki.name, "is", loki.animal_type, "but has", loki.characteristic)
print(loki.name, "is", loki.property) #accessing an instance attribute:
loki.property = "very cute" #changing it
print(loki.name, "is", loki.property)
loki_dict = loki.cat_info()
print("Dictionary about Loki:", loki_dict)
print("Type of the object loki:", type(loki))
print("Is loki an instance of Munchkin?")
print(isinstance(loki, Munchkin)) #using the isinstance function!
print("Is loki an instance of Cat?")
print(isinstance(loki, Cat)) #loki is an instance of Munchkin, but also of Cat
```