Introduction to Python

Objects 2: Inheritance and Hooks

Why Objects?

Object-oriented programming can be used for:

- Encapsulation
- Abstraction
- **Code reuse**

Inheritance

- Inheritance is a mechanism by which one class **inherits** code from another class.

Inheritance

- Inheritance is a mechanism by which one class **inherits** code from another class.
- Best seen by example. Lets write a program to model cars.

```
class Car():
    def changeGear(self, gear):
        self.gear = gear

    def printMe(self):
        print "I'm a car."
```

Inheritance

```
class Car():
    def changeGear(self, gear):
        self.gear = gear

    def printMe(self):
        print "I'm a car."

class Ford(Car):
    def printMe(self):
        print "I'm a Ford!"
x = Ford()
x.printMe() # prints "I'm a Ford!"
x.changeGear(5)
print x.gear #prints 5
```

How does inheritance work?

- Every class inherits from `object` implicitly.

How does inheritance work?

- Every class inherits from `object` implicitly.
- If `X` inherits from `Y` then we say `X` is a **subclass** of `Y`.

How does inheritance work?

- Every class inherits from `object` implicitly.
- If `X` inherits from `Y` then we say `X` is a **subclass** of `Y`.
- When you call a method on an instance the interpreter checks if the class has that method. If not it looks up the inheritance chain until it finds it.

Constructors of Subclasses

If you need to initialise data then you must tell your superclass to initialise itself as well.

```
class Dog():
    def __init__(self, colour):
        self.colour = colour

class BlackDog(Dog):
    def __init__(self, name):
        Dog.__init__(self, "Black")
        self.name = name
```

Make sure to initialise everything in the superclass before setting up the subclass.

What is a hook?

- Consider the following code:

```
print 1+2 #   3
print "h" + "i" # hi
print ["h"] + ["i"] # ["h","i"]
```

What is a hook?

- Consider the following code:

```
print 1+2 #  3
print "h" + "i" # hi
print ["h"] + ["i"] # ["h","i"]
```

- All of these types (`int`, `str`, `list`) know how to respond to `print` and +.

What is a hook?

- Consider the following code:

```
print 1+2 #  3
print "h" + "i" # hi
print ["h"] + ["i"] # ["h","i"]
```

- All of these types (`int`, `str`, `list`) know how to respond to `print` and +.
- Using `print` on our classes so far has resulted in fairly useless output.

What is a hook?

- Recall the `Rational` class from last time:

```
class Rational():
    def __init__(self, numerator, denominator):
        self.numerator = numerator
        self.denominator = denominator

half = Rational(1,2)
print half
```

Prints "__main__.Rational instance at 0x10a183440"

What is a hook?

- Recall the `Rational` class from last time:

```
class Rational():
    def __init__(self, numerator, denominator):
        self.numerator = numerator
        self.denominator = denominator

half = Rational(1,2)
print half
```

Prints "__main__.Rational instance at 0x10a183440"

- Because I haven't told `Rational` how to respond to print, it defaults to printing its location in memory.

What is a hook?

- Recall the `Rational` class from last time:

```
class Rational():
    def __init__(self, numerator, denominator):
        self.numerator = numerator
        self.denominator = denominator

half = Rational(1,2)
print half
```

Prints "__main__.Rational instance at 0x10a183440"

- Because I haven't told `Rational` how to respond to print, it defaults to printing its location in memory.
- Bonus question: how does it know how to do that?

What is a hook?

- Let's fix that behaviour:

```
class Rational():
    def __init__(self, numerator, denominator):
        self.numerator = numerator
        self.denominator = denominator

    def __repr__(self):
        return str(self.numerator) + '/' + str(self.d
```

What is a hook?

- Let's fix that behaviour:

```
class Rational():
    def __init__(self, numerator, denominator):
        self.numerator = numerator
        self.denominator = denominator

    def __repr__(self):
        return str(self.numerator) + '/' + str(self.d
```

- `__repr__` is a **hook** that the statement `print` calls on it's arguments.

What is a hook?

- Let's fix that behaviour:

```
class Rational():
    def __init__(self, numerator, denominator):
        self.numerator = numerator
        self.denominator = denominator

    def __repr__(self):
        return str(self.numerator) + '/' + str(self.d
```

- `__repr__` is a **hook** that the statement `print` calls on it's arguments.
- This is very useful for keeping track of what your program is doing!
- `__` is pronounced **dunder**- so dunder repr

Important Hooks

There are a ton of hooks that you can decide whether to implement for your class. Here are a few I find useful (see Python documentation for full list):

- `__eq__(self, other)`: How to respond to comparison with ==.
- `__add__(self,other)`: How to respond to +.
- `__getitem__(self,other)`: implement access to data using the `x[integer]` notation.

Use wisely. If you don't think hard about this you can really confuse your users (or yourself)!

Demo

Demo Time!