Lesson 6

Python Essentials

Overview

1. Classes
2. Inheritance
3. Mixins

Classes Overview

- Create reusable components
- Group data and operations together
- Classes are nouns
- Properties are adjectives
- Methods are verbs

Creating Classes

```
class Presenter():

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

    def say_hello(self):
        # method
        print('Hello, ' + self.name)

    @property
    def name(self):
        return self.__name

    @name.setter
    def name(self, value):
        # cool validation here
        self.__name = value
```

Using Classes

```
presenter = Presenter('Chris')
presenter.name = 'Christopher'
presenter.say_hello()
print(presenter.name)
```

Accessibility in Python

- **EVERYTHING is public**
- Conventions for suggesting accessibility
- _ means avoid unless you really know what you're doing
- __ (double underscore) means **do not use**

Inheritance

- Creates an "is a" relationship
  - Student is a Person
  - SqlConnection is a DatabaseConnection
  - MySqlConnection is a DatabaseConnection
- Composition (with properties) creates a "has a" relationship
  - Student has a Class
  - DatabaseConnection has a ConnectionString

Python Inheritance

- All methods are "virtual"
  - Can override or redefine their behavior
- Keyword super to access parent class
  - Constructor
  - Properties in methods
- Must always call parent constructor

Inheriting from a class

```
class Person:
    def __init__(self, name):
        self.name = name
    def say_hello(self):
        print('Hello, ' + self.name)

class Student(Person):
    def __init__(self, name, school):
        super().__init__(name)
        self.school = school
    def sing_school_song(self):
        print('Ode to ' + self.school)
```

Using a derived class

```
student = Student('John', 'Doe')
student.say_hello()
student.sing_school_song()

print(isinstance(student, Student))
print(isinstance(student, Person))
print(issubclass(Student, Person))
```

Mixins

- Inherit from multiple classes.
- A little controversial.
- Can get messy quickly.
- Many modern languages only support single inheritance.
- Uses:
    - Enable functionality for frameworks such as Django.
    - Streamline repetitious operations.

Using mixins

```
class Loggable:
    def __init__(self):
        self.title = ''
    def log(self):
        print('Log message from ' + self.title)
```

```python
class Connection:
    def __init__(self):
        self.server = ''
    def connect(self):
        print('Connecting to database on ' + self.server)

class SqlDatabase(Connection, Loggable):
    def __init__(self):
        super().__init__()
        self.title = 'Sql Connection Demo'
        self.server = 'Some_Server'

def framework(item):
    # Perform the connection
    if isinstance(item, Connection):
        item.connect()
    # Log the operation
    if isinstance(item, Loggable):
        item.log()

# Create an instance of our class
sql_connection = SqlDatabase()
# Use our framework
framework(sql_connection) # connects and logs
```

Notes:

- Create:
  - Helper database class
  - Create different types for different databases
- Function:
  - Connect to a database
  - Log what it's doing