Ch. 13: The Four Pillars of OOP

Four Pillars

- Encapsulation
- Abstraction
- Polymorphism
- Inheritance

Encapsulation

- Object variables and methods are located within the class the object is defined in
- This allows for clients using the code to create instances of the objects without modifying the actual variables and methods of the class
- In Python, there are no private variables or methods. You can only use naming conventions to warn the client to use at their own discretion

```
class PublicPrivateExample
    def __init__(self):
        self.public = 'safe'
        self._unsafe = 'unsafe'

    def public_method(self):
        # Clients can use This
        pass

    def _unsafe_method(self):
        # Clients shouldn't use this
        pass
```

Abstraction

- Process of taking away or removing characteristics from something in order to reduce it to a set of essential characteristics
- This occurs in OOP through creating objects from the Class models

Polymorphism

- The ability to present the same interface (function or method) for different underlying data types
- E.g. the print() function
- You can print multiple different data types to the console and the underlying method remains the same

```
print('Hello, World!')
print(200)
```

- The print function is an example of this. Different data types can be passed through the same function
- This is true of multiple built-in functions in Python

Inheritance

- Class inheritance is similar to biological inheritance
- A class can inherit attributes from another class
  - The class inheriting attributes is called the ***child class***
  - The classing whose attributes are being inherited is the ***parent class***
- Here is an example of inheritance using a Shape class

```
class Shape():
    def __init__(self, w, l):
        self.width = w
        self.length = l
    def print_size(self):
        print(%s by %s)
              % (self.width, self.length))

my_shape = Shape(20, 25)
```

- A child class can inherit the properties by taking the name of the parent class as a parameter
- You can use this to have general attributes and methods in the parent class, but more specific methods and attributes in the child
- For example, many shapes have similar attributes like length and width, but sometimes the area or volume are calculated differently
- You can use the child class to have a specific method, i.e. the calculation of a squares area

```
# Uses the Shape class created above as the parent
class Square(Shape):
```

```
        def area(self):
            return self.width * self.length
    square = Square(20, 20)
```

- As shown here, you use the same instantiation as the parent Shape class, but you include a specific method for a shape type
- You can also override parent methods in the child class. This is called ***method overriding***

Composition

- When you store an object as a variable in another object

Ch. 14: More OOP

Class variables vs Instance Variables

- Classes have two types of variables: ***class variables*** and ***instance variables***
- Instance variables are the attributes created when an instance of the class is created

```
class Shape():
    def __init__(self, w, l):
        # These are the instance variables
        self.width = w
        self.len = l
```

- Class variables belong to the class itself, not the instances of the class

```
class Shape():
    # This list of rectangle objects is a class variable
    recList = []
```

Magic Methods

- Every class in Python inherits from a parent class called ***Object***

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

lion = Lion('Dilbert')
print(lion)

>> <__main__.Lion object at 0x101178828>
```

- **init** is a ***magic method*** that initializes an instance of the class object
- Another default class magic method is **repr**, which is called when you print a class object
- Usually, it prints the type and location of the object, as in the example
- However, you can override magic methods

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

    def __repr__(self):
        return self.name

lion = Lion('Dilbert')
print(lion)

>> Dilbert
```

- Even simple addition, such as 2 + 2, invokes a magic method, **add**, in Python. This can also be overridden

```
class Add:
    def __init__(self, x):
        self.x = x

    def __add__(self, other):
        return self.x - other.x

a = Add(4)
b = Add(6)

print(a + b)

>> -2
```

Ch. 15: Bringing It All Together