Python Classes

Defining Python Class

- To define a Python class, you code the class definition that starts with the class keyword followed by the name of the class and a colon.
- It is common to start user-defined class names with an uppercase letter.
- A constructor is a special method named __init__ that defines the attributes for an object and initializes the values of those attributes.
- The constructor is automatically called whenever an object is created from the Python class.
- All methods including the constructor must take a reference to the object itself as their first parameters. By convention, this reference is named self.
- To make a private attribute, you can prefix the name of the attribute with a double underscores.
    - self.__title = title
- The __str__ method is useful for a string representation of the object, either when you can call str(your_object) to convert the object to an string, you can also call print(your_object) to print it out as a string.

Example of Book Class

bookstore.py

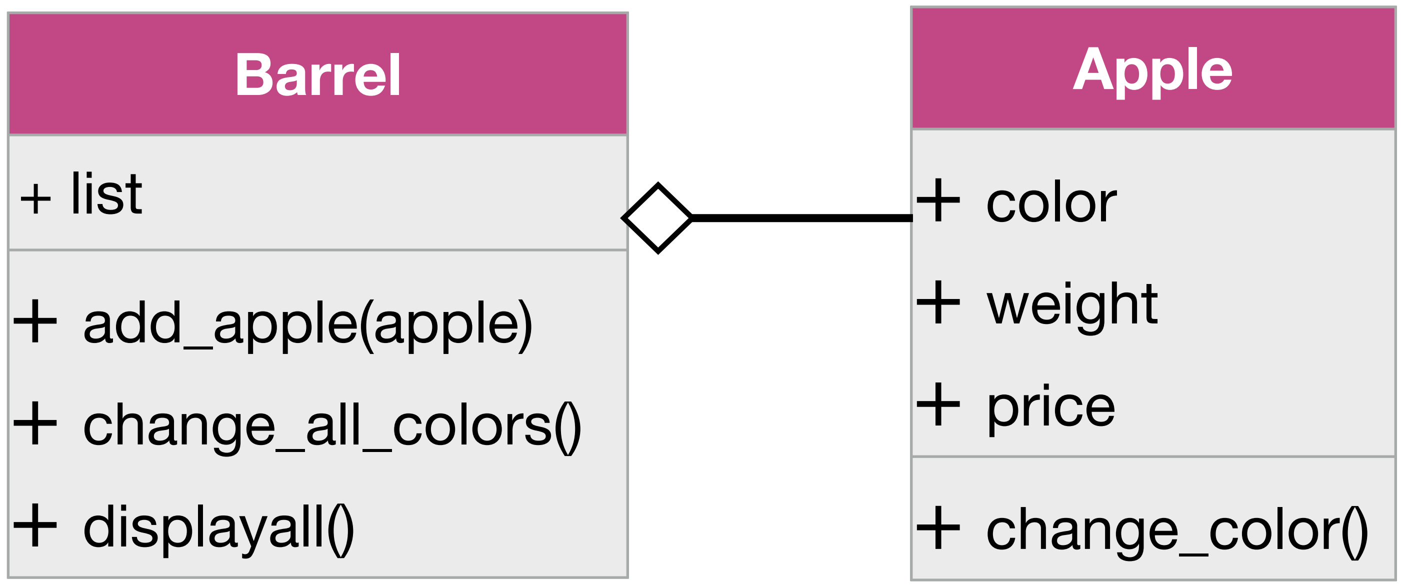
```
class Book:
    # a constructor that initializes 3 attributes
    def __init__(self, title, author="", price=0.0):
        self.__title = title
        self.__author = author
        self.price = price

    # a method that get the book title
    def get_title(self):
        return self.title

    # a method that sets the book title
    def set_title(self, title):
        self.__title = title

    # a method that get the book author
    def get_author(self):
        return self.author

    # a method that sets the book author
    def set_author(self, author):
        self.__author = author

    def __str__(self):
        return self.__title + ","  + str(self.__author) + "," + str(self.price)
```

use book.py

```
from bookstore import Book

b1 = Book('How to C++', 'Peter', 50)
b2 = Book('Python Programming')

print('b1 =', str(b1))
print('b2 =', str(b2))
print()
b2.set_author('Lily')
b2.price = 56.7
print("After b2.set_author('Lily') and b2.price = 56.7")
print('b2 =', str(b2))
print()
b2.title = 'Advanced C'
print("After b2.title = 'Advanced C'")
print('b2 =', str(b2))
print()
b2.set_title('Advanced C')
print("b2.set_title('Advanced C')'")
print('b2 =', str(b2))
```

Object Composition

- As we discussed before, composition / aggregation is a way to combine simple object into more complex objects. Example, a barrel of apples.



```
# composition.py
import random

colors = ["Green", "Red", "Yellow"]

class Apple:
    def __init__(self, color, weight, price):
        self.color = color
        self.weight = weight
        self.price = price

    def change_color(self):
        clr = random.randint(0,2)
        self.color = colors[clr]

    def __str__(self):
        return self.color + ', ' + "{:.2f}".format(self.weight) + ', ' +
"{:.2f}".format(self.price)


class Barrel:
    def __init__(self):
        self.list = []

    def add_apple(self, apple):
        self.list.append(apple)

    def change_all_colors(self):
        for apple in self.list:
            apple.change_color()

    def displayall(self):
        for apple in self.list:
            print(str(apple))

def main():
    a1 = Apple('Yellow', 0.5, 2.0)
    a2 = Apple('Green', 0.56, 2.5)
    a3 = Apple('Red', 1.2, 3.5)

    barrel = Barrel()
    barrel.add_apple(a1)
    barrel.add_apple(a2)
    barrel.add_apple(a3)
    barrel.displayall()

    print('\nAfter color changed:')
    barrel.change_all_colors()
    barrel.displayall()

if __name__ == "__main__":
    main()
```

```
Yellow, 0.50, 2.00
Green, 0.56, 2.50
Red, 1.20, 3.50

After color changed:
Yellow, 0.50, 2.00
Green, 0.56, 2.50
Green, 1.20, 3.50
```

Encapsulation

- Encapsulation allows us to hide the data attributes of an object from other code that uses the object. This is also called information hiding.
- Public attributes are can be accessed directly from the code that uses that object.
- Private attributes can only be accessed indirectly through public methods or properties in Python.
- In Python, to make a private attribute, prefix the name of the attribute with a double under store "__".
- If your code tries to access a private attribute, it causes an AttributeError.
- An interface allows a programmer to use an object in an abstract way without understanding its internal implementation. If the interface remains the same, we can change the internal implementation without changing other code that uses the object.

Setters and Getters

- A getter method (also known as accessor) is a method that gets the value of the attribute.
- A setter method (also known as mutator) is a method that sets the value of the attribute.
- By convention, getter and setter methods begin with get and set respectively.

```
# encapsulation1.py

class Apple:
    def __init__(self, color, weight, price):
        self.__color = color
        self.__weight = weight
        self.__price = price

    def set_price(self, price):
        self.__price = price

    def get_price(self):
        return self.__price

    def set_weight(self, weight):
        self.__weight = weight

    def get_weight(self):
        return self.__weight

    def set_color(self, color):
        self.__color = color

    def get_color(self):
        return self.__color

    def __str__(self):
        return self.__color + ', ' +
"{:.2f}".format(self.__weight) + ', ' +
"{:.2f}".format(self.__price)


def main():
    a1 = Apple('Yellow', 0.5, 2.0)
    print("a1 =", str(a1))

    a1.set_price(3.5)
    print("a1 =", str(a1))

    print(a1.__price)

if __name__ == "__main__":
    main()
```

```
a1 = Yellow, 0.50, 2.00
a1 = Yellow, 0.50, 3.50

Traceback (most recent call last):
    print(a1.__price)
AttributeError: 'Apple' object has no attribute '__price'
```

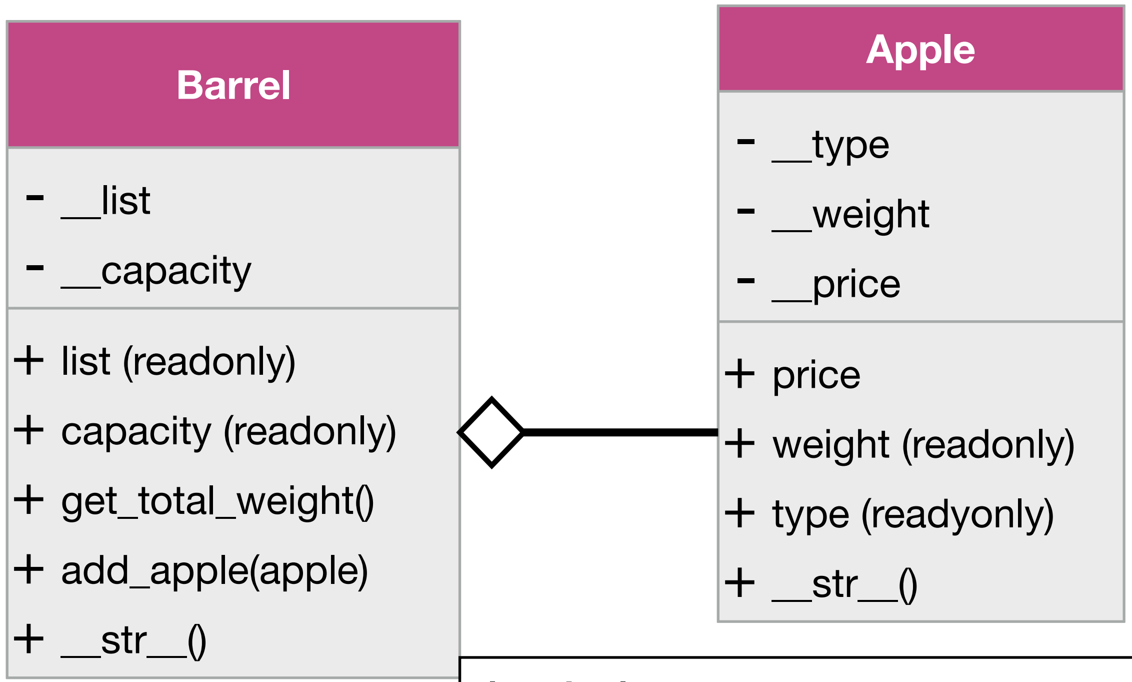
Using Properties

- In Python, you can also use a property to get and set a private attribute. A property is a special type of method.
- To code a property, use @property annotation above the method:
  - @property - for the getter method
  - @propertyName.setter - for the setter method.

```
# encapsulation2.py

class Apple:
    def __init__(self, color, weight, price):
        self.__color = color
        self.__weight = weight
        self.__price = price

    @property
    def price(self):
        return self.__price

    @price.setter
    def price(self, price):
        self.__price = price

    @property
    def weight(self):
        return self.__weight

    @weight.setter
    def weight(self, weight):
        self.__weight = weight

    @property
    def color(self):
        return self.__color

    @color.setter
    def color(self, color):
        self.__color = color

    def __str__(self):
        return self.__color + ', ' + "{:.2f}".format(self.__weight) + ', ' + "{:.2f}".format(self.__price)

def main():
    a1 = Apple('Yellow', 0.5, 2.0)
    print("a1 =", str(a1))

    a1.price = 3.5
    print("a1 =", str(a1))

    a1.weight = 2.0
    print("a1 =", str(a1))

    print("a1.color =", a1.color)

if __name__ == "__main__":
    main()
```

Case Study - Market For Selling Apples

- This market allows customer to buy barrels of apples.



```
class Apple:
    def __init__(self, type, weight, price):
        self.__type = type
        self.__weight = weight
        self.__price = price

    @property
    def price(self):
        return self.__price

    @price.setter
    def price(self, price):
        if price < 0:
            raise ValueError('Apple price must be greater than zero,')
        else:
            self.__price = price

    @property
    def weight(self):
        return self.__weight

    @property
    def type(self):
        return self.__typr

    def __str__(self):
        return self.__type + ', ' \
            + "{:.2f}".format(self.__weight) \
            + ', ' + "{:.2f}".format(self.__price)
```

```
class Barrel:
    def __init__(self, capacity):
        self.__list = []
        self.__capacity = capacity

    @property
    def list(self):
        return self.__list

    @property
    def capacity(self):
        return self.__capacity

    def get_total_weight(self):
        totalWeight = 0.0
        for apple in self.list:
            totalWeight += apple.weight;
        return totalWeight

    def add_apple(self, apple):
        # check if it is over weight
        if self.get_total_weight() + apple.weight > self.capacity:
            print("Error! Total weight exceeds the capacity.")
            return False

        self.list.append(apple)
        return True

    def __str__(self):
        output = 'capacity: ' + "{:.2f}".format(self.capacity) + '\n'
        for apple in self.list:
            output += str(apple) + '\n'
        return output
```

Case Study - Market For Selling Apples

```python
from market import Apple, Barrel

def main():
    print("The Market Test Program")
    print()

    while True:
        capacity = float(input("Enter the capacity of the barrel: "))
        barrel = Barrel(capacity)

        while True:
            type = input("Enter apple type: ")
            weight = float(input("Enter apple weight: "))
            price = float(input("Enter apple price: "))

            apple = Apple(type, weight, price)
            barrel.add_apple(apple)

            choice = input("Add more apples? (y/n): ")
            print()
            if choice != "y":
                print('Your barrel has these apples:')
                print(barrel)
                break

        choice = input("Get another barrel? (y/n): ")
        print()
        if choice != "y":
            print("Bye!")
            break


if __name__ == "__main__":
    main()
```

```
The Market Test Program

Enter the capacity of the barrel: 10
Enter apple type: Fuji
Enter apple weight: 1.5
Enter apple price: 2
Add more apples? (y/n): y

Enter apple type: Gala
Enter apple weight: 1
Enter apple price: 1.2
Add more apples? (y/n): y

Enter apple type: Gala
Enter apple weight: 0.8
Enter apple price: 1
Add more apples? (y/n): y

Enter apple type: Red
Enter apple weight: 1.5
Enter apple price: 1.25
Add more apples? (y/n): n

Your barrel has these apples:
capacity: 10.00
Fuji, 1.50, 2.00
Gala, 1.00, 1.20
Gala, 0.80, 1.00
Red, 1.50, 1.25

Get another barrel? (y/n): n

Bye!
>>>
```

Exercises

- Enhance the Barrel class by
  - adding a new method called get_barrel_price() that returns the total of all apple prices.
  - adding a new method called get_apples_by_type(type) that returns a list of apples whose types are the specified type.
  - adding a new method called get_heavy_apples() that returns a list of apples that weigh more than 1 lb.
  - adding a new method called remove_small_apples() that removes all apples that weigh less 0.5 lb