Classes and Objects Review

Python Classes and Objects

- **Class Definition**: A class is a blueprint for creating objects. It defines the behavior an object will have through its attributes and methods. Here is a basic example of a class definition in Python:

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

    def bark(self):
        print(f'{self.name.upper()} says woof woof!')
```

- **Creating Objects**: Objects are instances of a class. They are created by calling the class with the necessary arguments.

```
dog1 = Dog('Jack', 3)
dog2 = Dog('Thatcher', 5)

dog1.bark()  # JACK says woof woof!
dog2.bark()  # THATCHER says woof woof!
```

- **Calling methods with objects**: You can call methods on objects to perform actions or retrieve information.

```
objectName1.methodName()
objectName2.methodName()
```

- **Difference Between Class and Object**: A class is a reusable template, while an object is a specific instance of that class with actual data.

Attributes

- **Instance Attributes**: Defined in `__init__()` using `self`, and unique to each object.
- **Class Attributes**: Defined directly inside the class and shared by all instances.

```
class Dog:
    species = 'French Bulldog'  # Class attribute

    def __init__(self, name):
        self.name = name  # Instance attribute

print(Dog.species) # French Bulldog

jack = Dog('Jack')
print(jack.name)     # Jack
print(jack.species)  # French Bulldog
```

Methods

- **Methods**: Functions defined inside a class that operate on the object's attributes.

```
class Car:
    def __init__(self, color, model):
```

```
    def describe(self):
        return f'This car is a {self.color} {self.model}'

my_car_1 = Car('red', 'Tesla Model S')
print(my_car_1.describe())  # This car is a red Tesla Model S
```

- **Accessing Methods**: Call methods on objects using the dot notation. Here is an example of calling the `describe` method on two different car objects:

```
class Car:
 def __init__(self, color, model):
   self.color = color
   self.model = model

 def describe(self):
   return f'This car is a {self.color} {self.model}'

my_car_1 = Car('red', 'Tesla Model S')
my_car_2 = Car('green', 'Lamborghini Revuelto')

print(my_car_1.describe()) # Calling methods using the dot notation

print(my_car_2.describe()) # Calling methods using the dot notation
```

Dunder (Magic) Methods

- **Definition**: Special methods that start and end with a double underscore (e.g., `__init__`, `__len__`, `__str__`, `__eq__`). Python uses them internally for built-in operations.

```
class Book:
    def __init__(self, title, pages):
        self.title = title
        self.pages = pages

    def __len__(self):
        return self.pages

    def __str__(self):
        return f"'{self.title}' has {self.pages} pages"

    def __eq__(self, other):
        return self.pages == other.pages

book1 = Book('Built Wealth Like a Boss', 420)
print(len(book1))        # 420
print(str(book1))        # 'Built Wealth Like a Boss' has 420 pages
```

- **Calling dunder methods indirectly**: You don't need to call dunder methods directly. Instead, Python automatically calls them wher certain actions happen. These operations include:
    - **arithmetic operations like addition, subtraction, multiplication, division, and others**. In addition, `__add__()` is called, `__sub__()` for subtraction, `__mul__()` for multiplication, and `__truediv__()` for division.
    - **string operations like concatenation, repetition, formatting, and conversion to text**. `__add__()` is called for concatenation `__mul__()` for repetition, `__format__()` for formatting, `__str__()` and `__repr__()` for text conversion, and so on.

and `__next__()` to fetch the next item.

Real World Example: Shopping Cart

- **Cart Class with Dunder Methods**: Allows adding, removing, iterating, and checking contents with built-in behavior.

```
class Cart:
    def __init__(self):
        self.items = []

    def add(self, item):
        self.items.append(item)

    def remove(self, item):
        if item in self.items:
            self.items.remove(item)
        else:
            print(f'{item} is not in cart')

    def list_items(self):
        return self.items

    def __len__(self):
        return len(self.items)

    def __getitem__(self, index):
        return self.items[index]

    def __contains__(self, item):
        return item in self.items

    def __iter__(self):
        return iter(self.items)

cart = Cart()
cart.add('Laptop')
print(len(cart))        # 1
print('Laptop' in cart) # True
```

Assignment

- [ ] Review the Classes and Objects topics and concepts.

Please complete the assignment

Submit

Ask for Help