Classes

Classes and methods are fundamental concepts in object-oriented programming (OOP) and are a key feature of the Python language. Classes allow you to define your own data types and methods, which are functions that operate on those data types. Let's dive in!

Defining a Class

To define a class, we use the `class` keyword. The name of the class should be in `CamelCase` format, and the body of the class should be indented. For example:

```
class Dog:
    pass
```

In this example, we've defined a class called `Dog`. The `pass` statement is a placeholder that tells Python to do nothing. We'll fill in the class later.

Creating an Instance of a Class

Now we've designed a class, we can create an instance of that class. An instance is an object that is constructed from a class. For example:

```
my_dog = Dog()
```

In this example, we've created an instance of the `Dog` class and assigned it to the variable `my_dog`.

Attributes

Instance Attributes

Now that we have an object, we can assign instance attributes to that object. Instance attributes are variables that are specific to each instance of a class. For example:

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

my_dog = Dog('Rex', 2)
```

```
print(my_dog.name, my_dog.age)
```

In this example, we've defined an `__init__` method for our `Dog` class. `__init__` is a special method that gets called when an object is created. It takes the parameters `name` and `age` and assigns them to the instance attributes `self.name` and `self.age` respectively. The `self` parameter is a reference to the current instance of the class, and is used to access variables that belong to the class.

Class Attributes

Class variables are variables that are shared by all instances of a class. To define a class variable, we define a variable inside the class but outside of any methods. For example:

```
class Dog:
    num_legs = 4

    def __init__(self, name, age):
        self.name = name
        self.age = age

my_dog = Dog('Rex', 2)
print(my_dog.num_legs)
```

In this example, we've defined a class variable `num_legs` with a value of `4`. This is unaffected by our choice of input parameters when creating an instance of the class.

Methods

Methods are functions that are defined inside the body of a class using the `def` keyword. They are used to define the behaviors of the class. For example:

```
class Dog:
    num_legs = 4

    def __init__(self, name, age):
        self.name = name
        self.age = age

    def bark(self):
        print("Woof!")
```

```
my_dog = Dog('Rex', 2)
my_dog.bark()
```

In this example, we've defined a method called `bark` that takes the parameter `self`. The `self` parameter references the current instance of the class, and allows the method to access the instance variables that belong to the class. The `bark` method prints the string `"Woof!"` to the console.

Inheritance

Class inheritance is a mechanism that allows you to create a new class that is a modified version of an existing class. The new class, known as the subclass, inherits attributes and behaviors from the existing class, known as the superclass. In Python, you can create a subclass by using the class keyword and specifying the superclass in parentheses after the subclass name. Here's an example of how to create a subclass:

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

    def speak(self):
        print("Hello!")

class Dog(Animal):
    def __init__(self, name, breed):
        super().__init__(name, species='dog')
        self.breed = breed

    def speak(self):
        print("Woof!")
```

In this example, we define an `Animal` class with instance variables name and species. We then define a `Dog` subclass that inherits from `Animal` and has an additional instance variable `breed`. In the Dog class's `__init__` method, we use `super()` to call the `__init__` method of the `Animal` superclass, passing in the name argument and setting the species argument to `'dog'`.

Overriding Methods and Attributes

In the above example, we defined a `Dog` class that inherits from the `Animal` class. We have overridden the attributes `name` and `species` from the `Animal` class, and we have overridden the `__init__` method from the `Animal` class. We have also added a new attribute `breed` and overridden the `speak` method. For example:

```
human = Animal('John', 'human')
print(human.name, human.species)
human.speak()
```

```
John human
Hello!
```

```
dog = Dog('Rex', 'lab')
print(dog.name, dog.species, dog.breed)
dog.speak()
```

```
Rex dog lab
Woof!
```

Exercises

1. Define a class `Rectangle` with instance variables `length` and `width`, and a method `area` that returns the area of the rectangle.
2. Define a class `Circle` with instance variable `radius`, and methods `area` and `circumference` that return the area and circumference of the circle, respectively.
3. Define a class `Person` with instance variables `name` and `age`, and a method `greet` that prints a greeting message with the person's name.
4. Define a class `Student` that inherits from `Person`, with an additional instance variable `major`, and a method `study` that prints a message that the student is studying their major.
5. Define a class `Car` with instance variables `make`, `model`, and `year`, and a method `get_age` that returns the age of the car in years (based on the current year).
6. Define a class `Employee` with instance variables `name`, `salary`, and `bonus`, and a method `total_pay` that returns the total pay (salary plus bonus) for the employee.

7. Define a class `BankAccount` with instance variables balance and `interest_rate`, and methods `deposit` and `withdraw` that modify the balance, and a method `add_interest` that adds interest to the balance based on the interest rate.
8. Define a class `ShoppingCart` with a list instance variable `items`, and methods `add_item` and `remove_item` that add and remove items from the list, and a method `total_cost` that returns the total cost of all the items in the list.
9. Define a `Person` class with instance variables `name` and `age` and a `greet` method that prints a greeting. Then define an `Employee` subclass that inherits from `Person` and adds an instance variable `salary` and a `work` method that prints a message indicating the employee is working. You can use your code from Exercise 3 as a starting point for your subclass.
10. Using the `BankAccount` class from Exercise 7, define two subclasses, `CheckingAccount` and `SavingsAccount`, that inherit from `BankAccount` and add instance variables `overdraft_fee` for checking accounts and `minimum_balance` for savings accounts. Implement the `withdraw` method in each subclass to include any additional fees or restrictions.

Project: Car Rental System

Create a Python program that simulates a car rental system. The program should use a class to define a car, and allow the user to rent a car and return it.

Project Requirements

1. The program should define a class named Car with methods to rent and return a car.
2. The program should handle errors appropriately, for example, when the user attempts to rent a car that is already rented.

Example Output

```
Car Rental System

Available cars:
1. Toyota Corolla
2. Honda Civic
3. Mazda3

Choose a car to rent (1/2/3): 2
```

```
Rented car: Honda Civic.

Available cars:
1. Toyota Corolla
3. Mazda3

Return the rented car (y/n)? y

Car returned. Thank you for renting from us!
```

Further Reading

Check out these following resources for more information on classes and methods:

- W3Schools
  - Python Classes and Objects
- RealPython
  - Python Classes and Objects
- Python Docs
  - Classes
  - Data Model
  - Special method names

You can also use the search terms "python classes" or "python methods" to find more resources online.