bytewise.

Bytewise Fellowship Program

DATA SCIENCE

Task #6

BWT- Data Science (Group1)

Submitted to: Mahrukh Khan
Submitted by: Usama Malik

Task: Modular Programming Approach in python (classes, inheritance , encapsulation - ch9 onward)

Class

A class is a blueprint for creating objects. It defines a set of attributes and methods that the objects created from the class will have.

Example:

```
class Dog:
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def bark(self):
        return "Woof!"
# Creating an object (instance) from the class
my_dog = Dog("Buddy", 5)
# Accessing attributes and methods
print(my_dog.name)  # Output: Buddy
print(my_dog.age)   # Output: 5
print(my_dog.bark()) # Output: Woof!
```

The __init__() Method

The __init__() method is a special function in a class that runs when you create a new object. It's used to set up the object with initial values.

```
class Dog:
    def __init__(self, name):
        self.name = name
my_dog = Dog("Buddy")
print(my_dog.name)  # Output: Buddy
```

Making an Instance from a Class

To make an instance, you call the class as if it were a function. This creates an object based on the class blueprint.

```
class Dog:
    def __init__(self, name):
        self.name = name
my_dog = Dog("Buddy")
print(my_dog.name)  # Output: Buddy
```

Calling Methods

Methods are functions defined inside a class. You call them using the instance of the class.

```
class Dog:
    def __init__(self, name):
        self.name = name
    def bark(self):
        return "Woof!"
my_dog = Dog("Buddy")
print(my_dog.bark())  # Output: Woof!
```

Inheritance

Inheritance lets you create a new class based on an existing class. The new class (child) inherits attributes and methods from the old class (parent).

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

```
    return "Woof!"
my_dog = Dog("Buddy")
print(my_dog.name)  # Output: Buddy
print(my_dog.bark())  # Output: Woof!
```

Importing Classes

You can use classes from other files by importing them. This helps organize code better.

```
# In dog.py
class Dog:
    def __init__(self, name):
        self.name = name
# In main.py
from dog import Dog
my_dog = Dog("Buddy")
print(my_dog.name)  # Output: Buddy
```

Reading from a File

You can read the contents of a file using the open function.

```
with open("example.txt", "r") as file:
    content = file.read()
    print(content)
```

File Paths

File paths specify the location of a file. They can be absolute (full path) or relative (based on current directory).

```
# Absolute path
"/home/user/documents/example.txt"
# Relative path
"documents/example.txt"
```

Writing to a File

You can write data to a file using the open function with the write mode ("w").

```
with open("example.txt", "w") as file:
    file.write("Hello, World!")
```

Exceptions

Exceptions are errors that happen during execution. You can handle them using try and except.

```
try:
    result = 10 / 0
except ZeroDivisionError:
    print("You can't divide by zero!")
```

Using json.dump() and json.load()

json.dump() writes Python objects to a JSON file. json.load() reads JSON data back into Python objects.

```
import json

data = {"name": "Buddy", "age": 5}

# Writing to a file
with open("data.json", "w") as file:
    json.dump(data, file)
```

```
# Reading from a file
with open("data.json", "r") as file:
    data = json.load(file)
    print(data)  # Output: {'name': 'Buddy', 'age': 5}
```

Summary

Classes help you create objects. Methods are functions inside classes. Inheritance lets you reuse code. You can import classes, read/write files, and handle errors. JSON functions help save/load data.