Model Class

What is it

- Describes real world entities
- Can be directly compared to variabels as it is used to store and pass data
- We can create data types and data structures out of model classes

Implementation

- Attributes are the most important part of model classes
- Methods are only used to work on the attributes of the class

Uses

- We can use them to easily pass data
    - Instead of passing multiple variable to every function that wants to perform similar operations, we can use a single class object to pass this information
- We can easily deserialize values from csv files and json files and have a bit of type safety
    - We know that the application will either crash or create default values if the file has missing data
- Custom Data structures
    - We can create our own data structures that better model how we want to work with our data if predefined data structures don't meet our needs

Creating a model class

- Attributes are the most important part
- Methods should directly work with attributes
- We can use access modifiers and methods to have better control of the data being writtent to attributes

1. Creating a basic model that does not have any methods and just contains attributes that need to be directly assigned

```
# Most basic model class
class Person():
    # None denotes that there is no value being stored at the moment
    name = None
    surname = None
    age = None

# Using the class
person = Person()
print("Before setting name:", person.name)

# Set the name
person.name = "John"
print("After setting name:", person.name)
```

```
Before setting name: None
After setting name: John
```

2. Passing the attributes as parameters on initialization so that we do not need to manually fill each value when creating the object

```
# Adding a constructor

class Person:
    def __init__(self, name, surname, age):
        self.name = name
        self.age = age
        self.surname = surname

# Create person
person = Person('Jack', 'Jackson', 30)

print(person.name)
```

```
Jack
```

3. Using access modifiers and methods to change the values that are being store

In this example, we only want the age to be incremented by 1, so we make the age attribute private to prevent the user from manually setting the age variable

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

        self.__age = age

    def incrememnt_age(self):
        self.__age += 1

    def get_age(self):
        return self.__age


# Create person
person = Person('Jack', 'Jackson', 30)

# Person original age
print("Original age", person.get_age())

# update age
person.incrememnt_age()
print("Updated age", person.get_age())
```

```
Original age 30
Updated age 31
```

```
# A more advanced implementation of access modifiers
class Person():
    def __init__(self, name, surname, age):
        self.name = name
        self.surname = surname

        self.__age = age

    def incrememnt_age(self):
        self.__age += 1
```