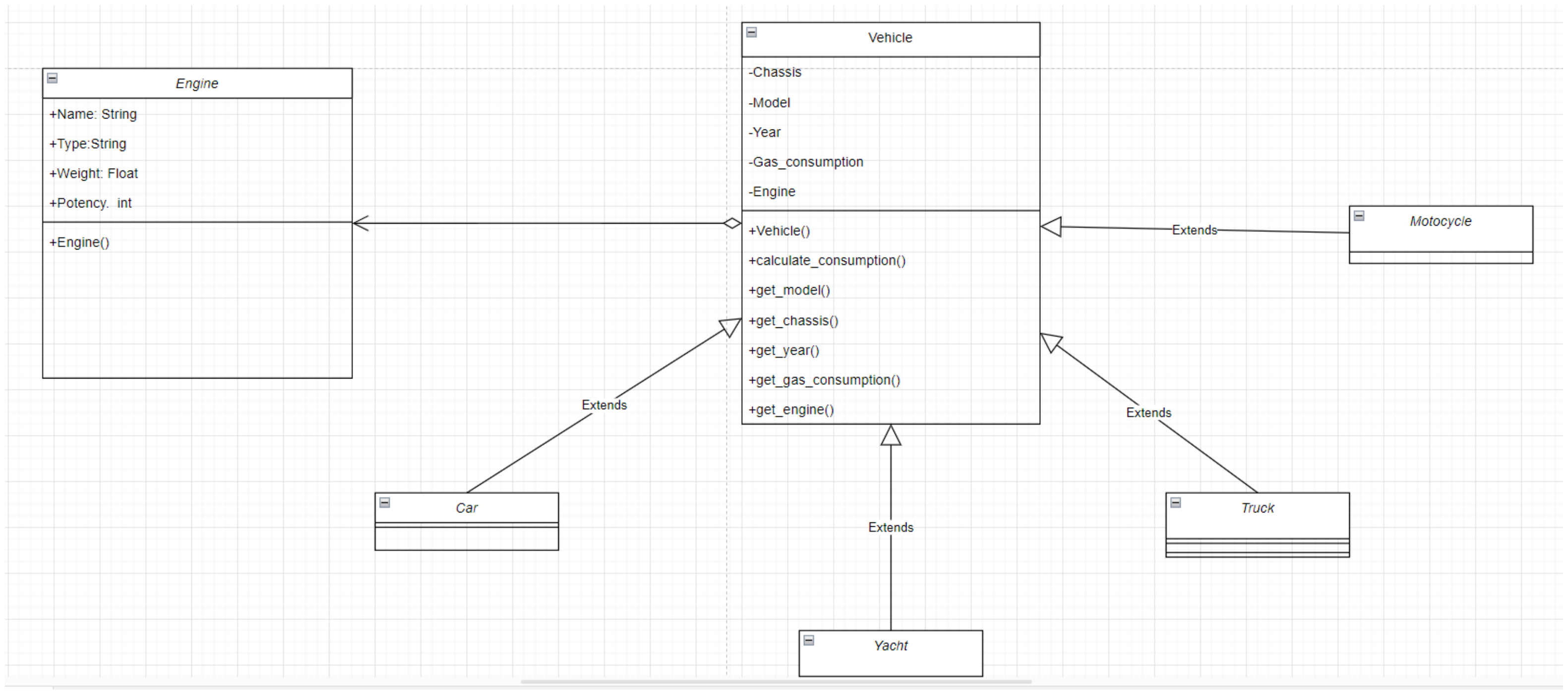
Technical report: First workshop advanced programming

Name: Carlos Alberto Barriga Gámez

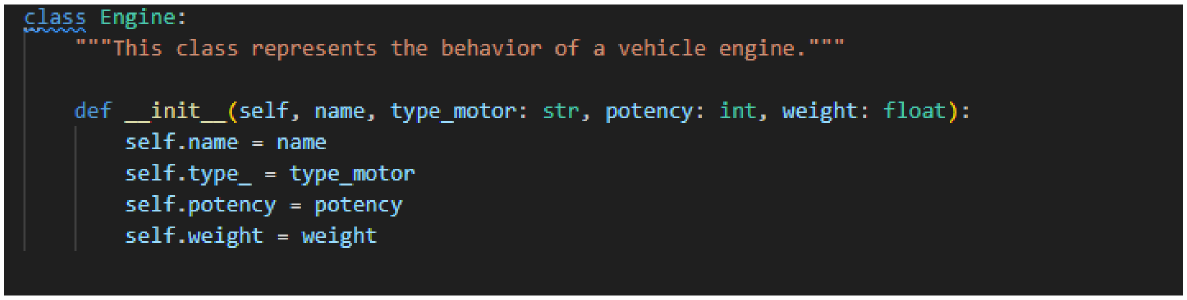
Code: 202220202179

Class diagram of the workshop:



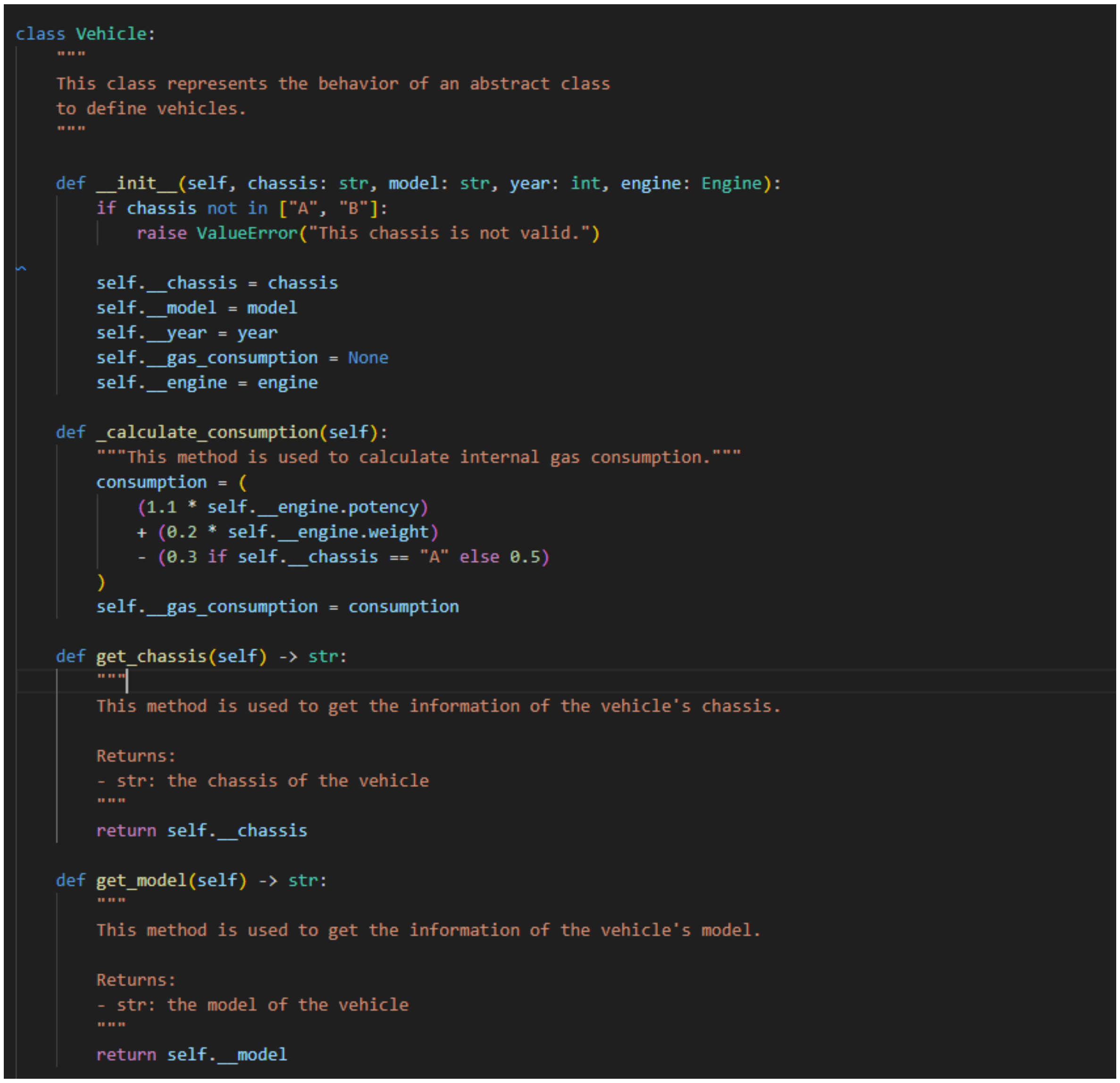
Technical decisions:

For the development of the vehicle company program, an engine object was created because it was necessary to have the ability to use its attributes, which is why name, type, power and weight were placed as parameters of the object, since these same attributes had values from the beginning.

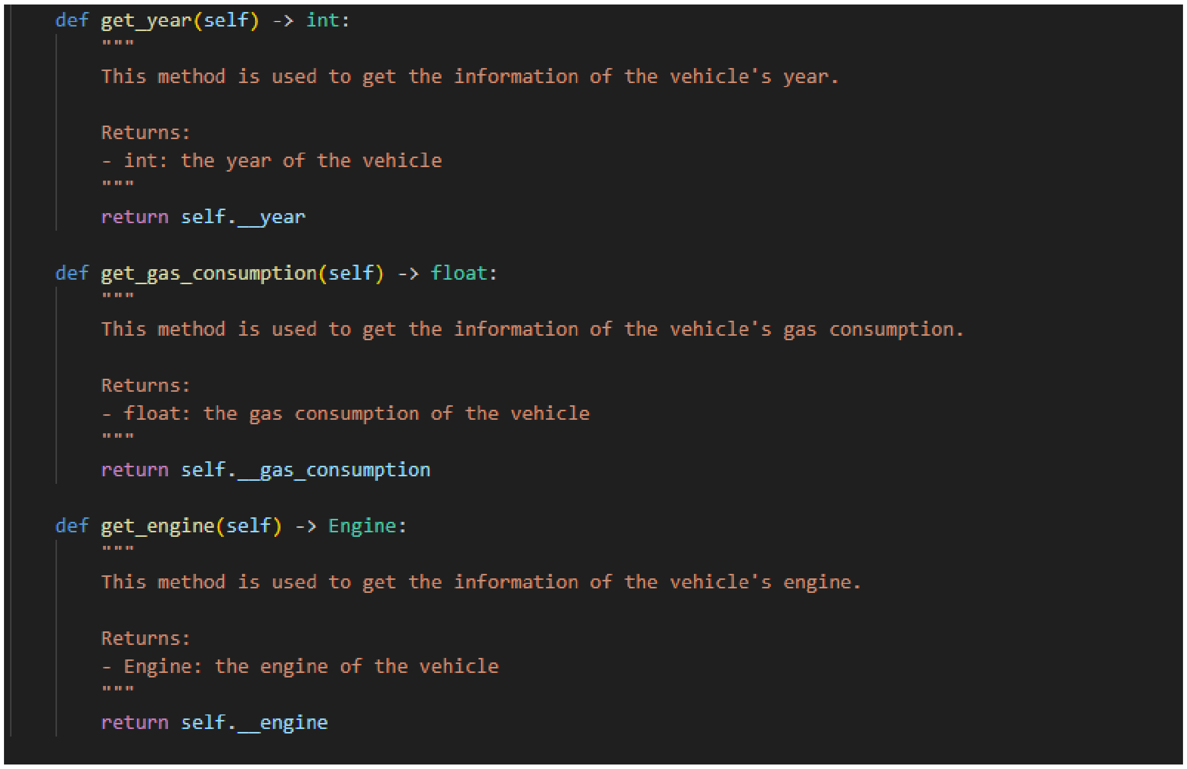


```python
class Engine:
    """This class represents the behavior of a vehicle engine."""

    def __init__(self, name, type_motor: str, potency: int, weight: float):
        self.name = name
        self.type_ = type_motor
        self.potency = potency
        self.weight = weight
```

Then a vehicle class was created with its attributes: chassis (with an exception that it could only be called A or B), model, year, engine and gas consumption, and again these attributes were placed in the object parameters. Also in this class, a method called "_calculate_consumption" was created, where the equation assigned in the requirements was operated, which in part depended on the weight and power of the engine, in addition to the type of vehicle chassis. Because the attributes of the vehicle class were created private, it was necessary to create get methods to obtain the information stored in them from outside the class.

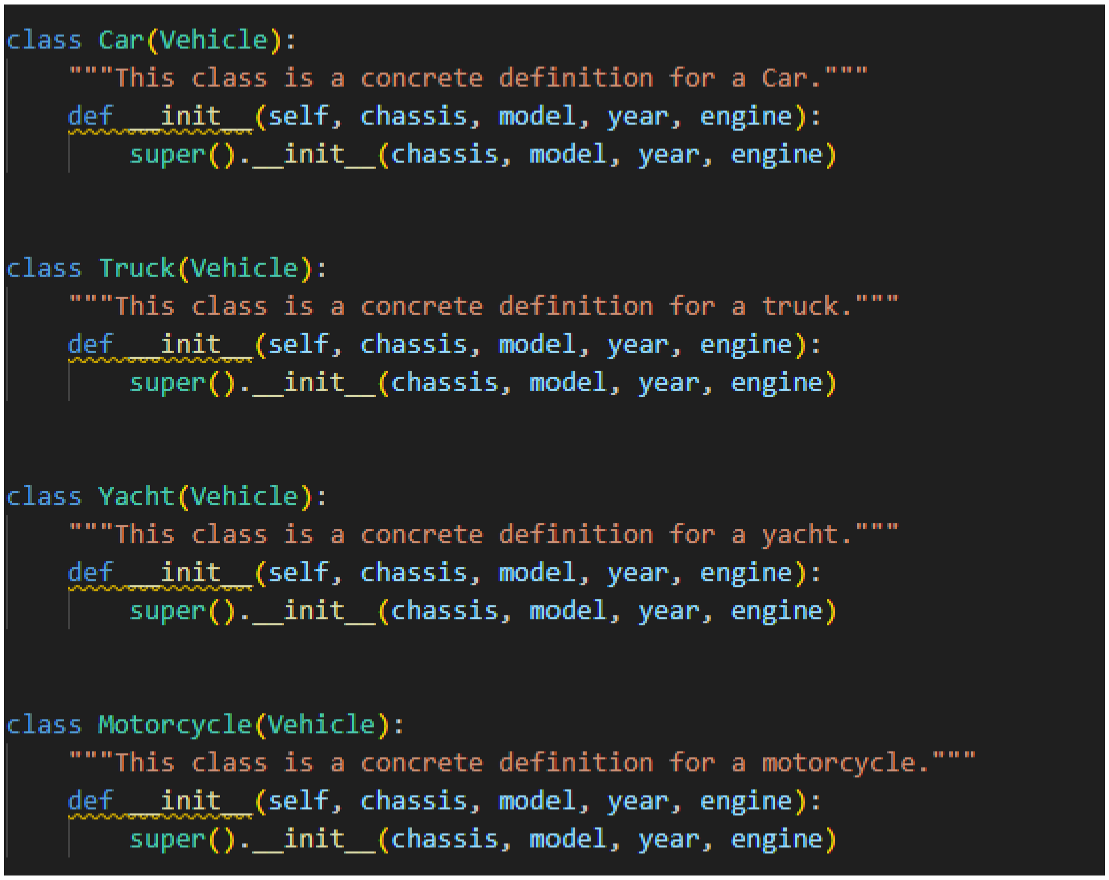


```python
class Vehicle:
    """
    This class represents the behavior of an abstract class
    to define vehicles.
    """

    def __init__(self, chassis: str, model: str, year: int, engine: Engine):
        if chassis not in ["A", "B"]:
            raise ValueError("This chassis is not valid.")

        self.__chassis = chassis
        self.__model = model
        self.__year = year
        self.__gas_consumption = None
        self.__engine = engine

    def _calculate_consumption(self):
        """This method is used to calculate internal gas consumption."""
        consumption = (
            (1.1 * self.__engine.potency)
            + (0.2 * self.__engine.weight)
            - (0.3 if self.__chassis == "A" else 0.5)
        )
        self.__gas_consumption = consumption

    def get_chassis(self) -> str:
        """
        This method is used to get the information of the vehicle's chassis.

        Returns:
        - str: the chassis of the vehicle
        """
        return self.__chassis

    def get_model(self) -> str:
        """
        This method is used to get the information of the vehicle's model.

        Returns:
        - str: the model of the vehicle
        """
        return self.__model
```



```python
    def get_year(self) -> int:
        """
        This method is used to get the information of the vehicle's year.

        Returns:
        - int: the year of the vehicle
        """
        return self.__year

    def get_gas_consumption(self) -> float:
        """
        This method is used to get the information of the vehicle's gas consumption.

        Returns:
        - float: the gas consumption of the vehicle
        """
        return self.__gas_consumption

    def get_engine(self) -> Engine:
        """
        This method is used to get the information of the vehicle's engine.

        Returns:
        - Engine: the engine of the vehicle
        """
        return self.__engine
```

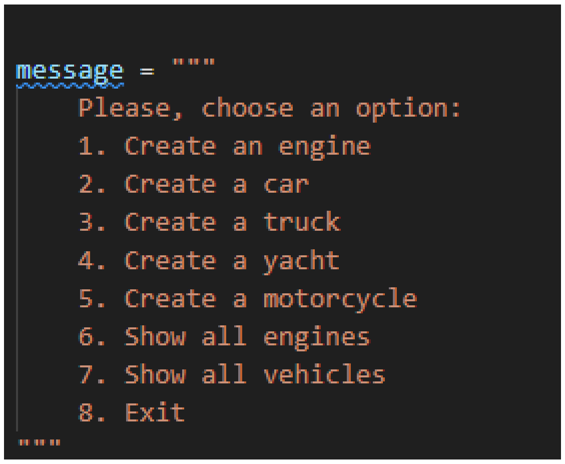
Also, as the statement said that there were different vehicles, it was decided to create these vehicles (car, truck, motorcycle and yacht), and the methods and attributes of the vehicle class were inherited to each of these.



```python
class Car(Vehicle):
    """This class is a concrete definition for a Car."""
    def __init__(self, chassis, model, year, engine):
        super().__init__(chassis, model, year, engine)


class Truck(Vehicle):
    """This class is a concrete definition for a truck."""
    def __init__(self, chassis, model, year, engine):
        super().__init__(chassis, model, year, engine)


class Yacht(Vehicle):
    """This class is a concrete definition for a yacht."""
    def __init__(self, chassis, model, year, engine):
        super().__init__(chassis, model, year, engine)


class Motorcycle(Vehicle):
    """This class is a concrete definition for a motorcycle."""
    def __init__(self, chassis, model, year, engine):
        super().__init__(chassis, model, year, engine)
```

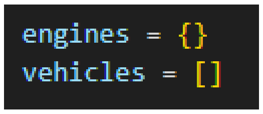
With the above, the initial requirements of the program were completed.

Now, as a requirement requested to create a menu where the user could create engines, vehicles and observe the existing engines and vehicles, it was decided to initially create a message with which the user was asked to choose an option:



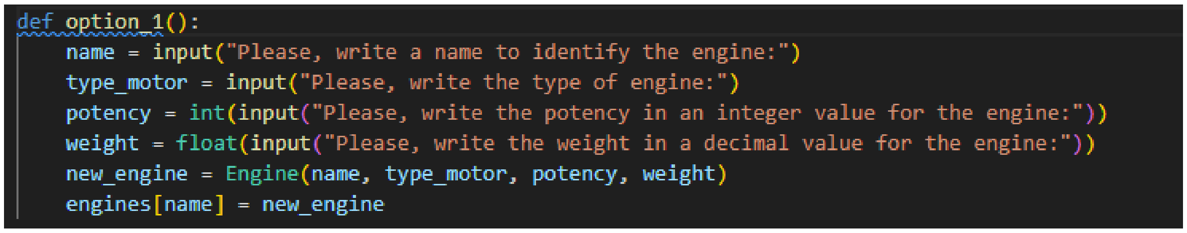
```python
message = """
    Please, choose an option:
    1. Create an engine
    2. Create a car
    3. Create a truck
    4. Create a yacht
    5. Create a motorcycle
    6. Show all engines
    7. Show all vehicles
    8. Exit
"""
```

Then a dictionary was created for the engines that the user registered, and a list to save the vehicles that the user created with their respective attributes.



```python
engines = {}
vehicles = []
```

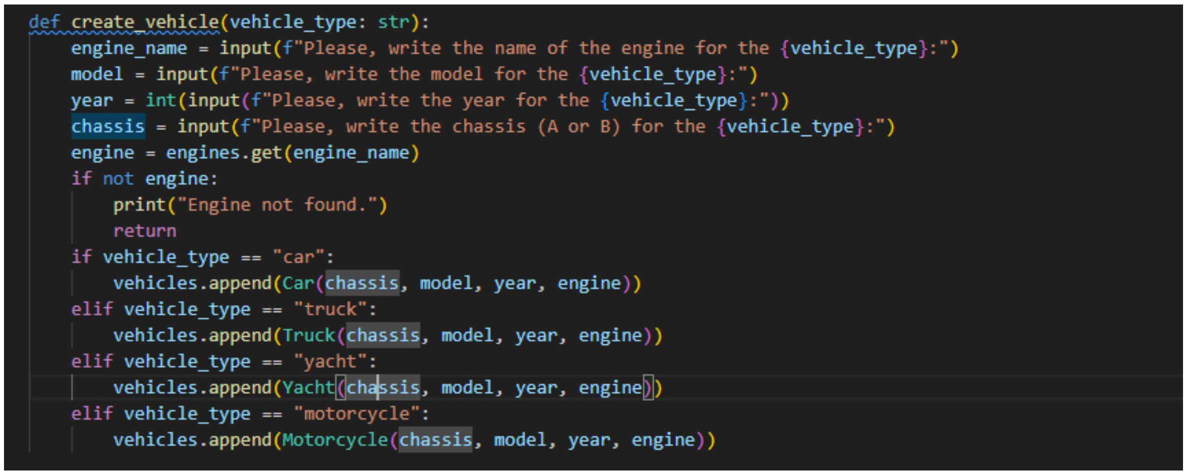
Later, to continue creating the menu that the user sees, a function called option_1 was created, which asks the user for the name, type of engine, power and weight of the engine, and with this data it creates a new engine, and stores it in the vehicle list.



```python
def option_1():
    name = input("Please, write a name to identify the engine:")
    type_motor = input("Please, write the type of engine:")
    potency = int(input("Please, write the potency in an integer value for the engine:"))
    weight = float(input("Please, write the weight in a decimal value for the engine:"))
    new_engine = Engine(name, type_motor, potency, weight)
    engines[name] = new_engine
```

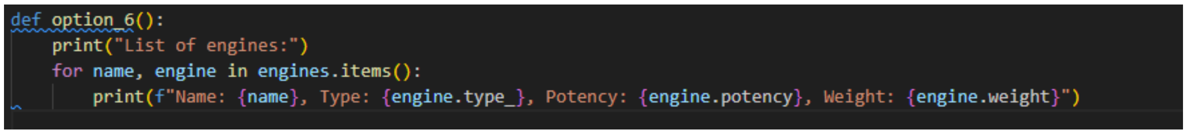
Then a function called create_vehicle was created with which the user is asked for information on the engine, the model, the year of release of the vehicle, the chassis and the engine. An if was also placed to check that the engine existed before; if not, the message "Engine not found" is displayed.

In the same function, an if statement was created where depending on the type of vehicle written by the user, that type of vehicle with its attributes was added to the vehicle list.



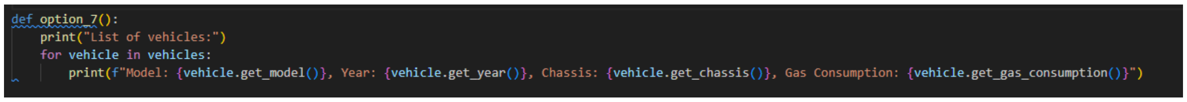
```python
def create_vehicle(vehicle_type: str):
    engine_name = input(f"Please, write the name of the engine for the {vehicle_type}:")
    model = input(f"Please, write the model for the {vehicle_type}:")
    year = int(input(f"Please, write the year for the {vehicle_type}:"))
    chassis = input(f"Please, write the chassis (A or B) for the {vehicle_type}:")
    engine = engines.get(engine_name)
    if not engine:
        print("Engine not found.")
        return
    if vehicle_type == "car":
        vehicles.append(Car(chassis, model, year, engine))
    elif vehicle_type == "truck":
        vehicles.append(Truck(chassis, model, year, engine))
    elif vehicle_type == "yacht":
        vehicles.append(Yacht(chassis, model, year, engine))
    elif vehicle_type == "motorcycle":
        vehicles.append(Motorcycle(chassis, model, year, engine))
```

Then a function called option_6 was created that is responsible for displaying the name, motor type, weight and power of the motors saved in the motor dictionary.



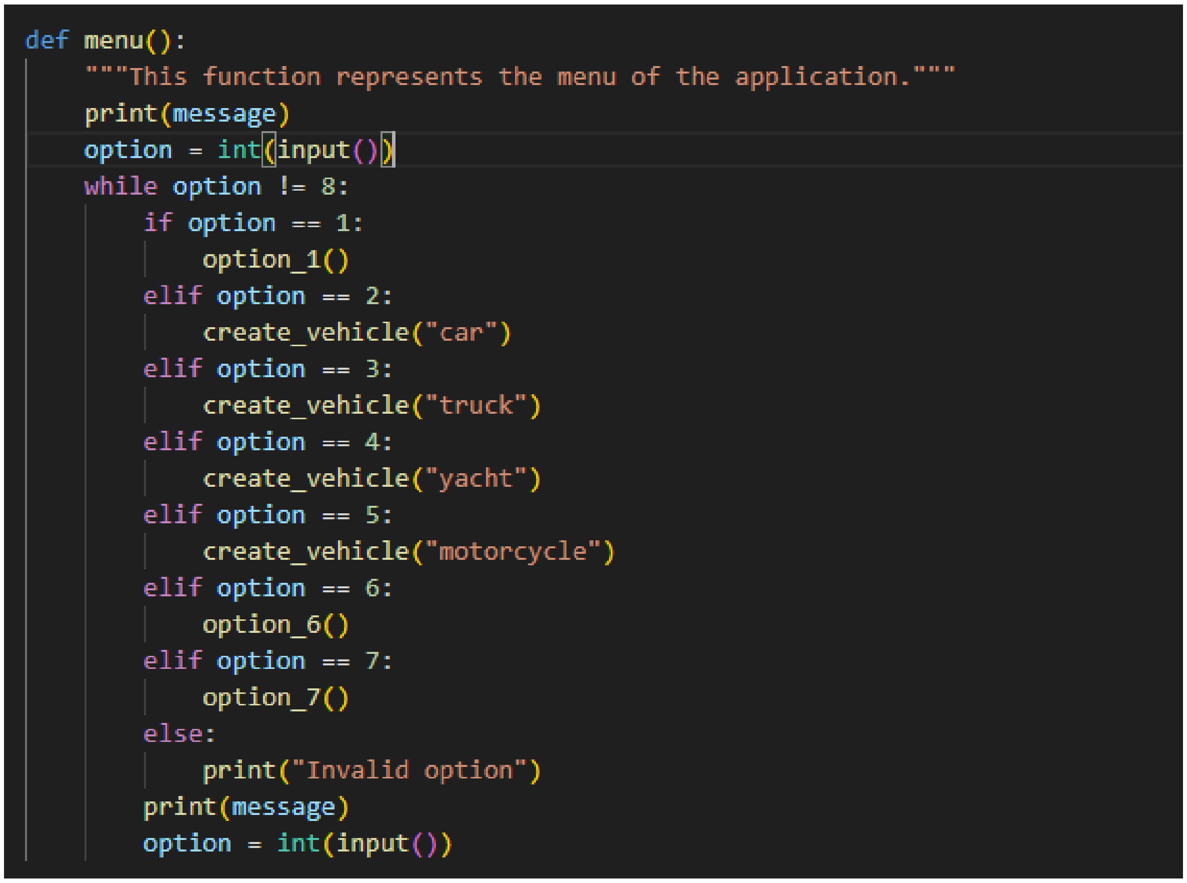
```python
def option_6():
    print("List of engines:")
    for name, engine in engines.items():
        print(f"Name: {name}, Type: {engine.type_}, Potency: {engine.potency}, Weight: {engine.weight}")
```

A function called option_7 allows displaying the vehicles (with their attributes) stored in the vehicle list.



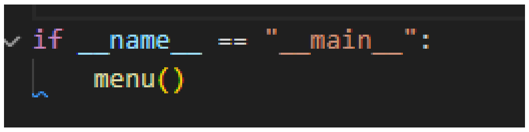
```python
def option_7():
    print("List of vehicles:")
    for vehicle in vehicles:
        print(f"Model: {vehicle.get_model()}, Year: {vehicle.get_year()}, Chassis: {vehicle.get_chassis()}, Gas Consumption: {vehicle.get_gas_consumption()}")
```

Then a menu function was created that solved the last requirement of the exercise. This showed the message created previously for the user to enter the number of the option that he will need, in this way option 1 allows creating an engine, options 2, 3, 4, 5, allow creating the respective vehicles according to the type of vehicle that the user wants, option 6 allows you to see the engines (with attributes saved in the dictionary) and option 7 allows you to see the vehicles along with their attributes. Finally, if the user enters the number 8, he exits the program.



```python
def menu():
    """This function represents the menu of the application."""
    print(message)
    option = int(input())
    while option != 8:
        if option == 1:
            option_1()
        elif option == 2:
            create_vehicle("car")
        elif option == 3:
            create_vehicle("truck")
        elif option == 4:
            create_vehicle("yacht")
        elif option == 5:
            create_vehicle("motorcycle")
        elif option == 6:
            option_6()
        elif option == 7:
            option_7()
        else:
            print("Invalid option")
        print(message)
        option = int(input())
```

Finally, an if conditional was created so that the menu function would be started in the startup class and the user would interact with it.



```python
if __name__ == "__main__":
    menu()
```