# Exercises: Inheritance

Problems for exercise and homework for the [Python OOP Course @SoftUni](https://softuni.bg/courses/python-oop).

Submit your solutions in the SoftUni judge system at <https://judge.softuni.bg/Contests/1941>.

## Person

You are asked to model an application for storing data about people. You should be able to have a Person and a Child. The child derives from the person. Every person receives **name** and **age** upon initialization. Your task is to model the application.

Create a **Child** class that inherits **Person** and has the same constructor definition. However, do not copy the code from the **Person** class - **reuse the Person class's constructor**.

Submit in judge a **zip file** named **project**, containing a separate file (person.py and child.py) for each of the classes.

|  |  |
| --- | --- |
| **Test Code** | **Output** |
| person = Person("Peter", 25)  child = Child("Peter Junior", 5)  print(person.name)  print(person.age)  print(child.\_\_class\_\_.\_\_bases\_\_[0].\_\_name\_\_) | Peter  25  Person |

## Zoo

Create a zoo which contains the following classes:

Submit in judge a **zip file** of the **project**, containing a separate file for each of the classes using the structure shown below:



Follow the diagram and create all the classes. **Each** of them, except the **Animal** class, should **inherit** from **another** **class** as shown in the diagram. The **Animal** class should receive a name - string upon initialization.

Every class should have constructor, which accepts one parameter: **name**

|  |  |
| --- | --- |
| **Test Code** | **Output** |
| mammal = Mammal("Stella")  print(mammal.\_\_class\_\_.\_\_bases\_\_[0].\_\_name\_\_)  print(mammal.name)  lizard = Lizard("John")  print(lizard.\_\_class\_\_.\_\_bases\_\_[0].\_\_name\_\_)  print(lizard.name) | Animal  Stella  Reptile  John |

## Players and Monsters

Your task is to create the following game hierarchy:

Submit in judge a **zip file**, containing a separate file for each of the classes using the structure shown below:



Create a class **Hero**. It should contain the following attributes:

* **username - string**
* **level - int**

Override the **\_\_str\_\_()** method of the base class so it returns: **"{name} of type {class\_name} has level {level}"**

|  |  |
| --- | --- |
| **Test Code** | **Output** |
| hero = Hero("H", 4)  print(hero.username)  print(hero.level)  print(str(hero))  elf = Elf("E", 4)  print(str(elf))  print(elf.\_\_class\_\_.\_\_bases\_\_[0].\_\_name\_\_)  print(elf.username)  print(elf.level) | H  4  H of type Hero has level 4  E of type Elf has level 4  Hero  E  4 |

## Need for Speed

Create the following **hierarchy** with the following **classes**:



and submit in judge a **zip file**, containing a separate file for each of the classes using the structure shown below:



Create a base class Vehicle. It should contain the following attributes:

* **DEFAULT\_FUEL\_CONSUMPTION: float (constant)**
* **fuel\_consumption: float – the given fuel consumption is per kilometer**
* **fuel: float – represent the fuel in the specific vehicle**
* **horse\_power: int**

The class should recieve fuel and horse\_power upon initialization and should **set** the **default fuel consumption** on the attribute fuel\_consumption.

The class should have the following methods:

* **drive(kilometers)** -reduces the **fuel** based on the travelled kilometers and fuel consumption. Keep in mind that you can drive the vehicle only if you have enough fuel to finish the driving.

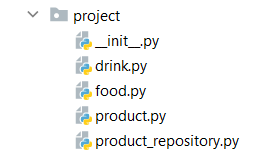
The default fuel consumption for **Vehicle** is 1.25.Some of the classes have **different** **default fuel consumption**:

* **SportCar - 10**
* **RaceMotorcycle - 8**
* **Car - 3**

|  |  |
| --- | --- |
| **Test Code** | **Output** |
| vehicle = Vehicle(50, 150)  print(Vehicle.DEFAULT\_FUEL\_CONSUMPTION)  print(vehicle.fuel)  print(vehicle.horse\_power)  print(vehicle.fuel\_consumption)  vehicle.drive(100)  print(vehicle.fuel)  family\_car = FamilyCar(150, 150)  family\_car.drive(50)  print(family\_car.fuel)  family\_car.drive(50)  print(family\_car.fuel)  print(family\_car.\_\_class\_\_.\_\_bases\_\_[0].\_\_name\_\_) | 1.25  50  150  1.25  50  0  0  Car |

## Shop

*Maria is expanding her business and today she is opening a grocery shop. You are hired to write a program which keeps track of the shop's inventory.*

******

In the **product.py** file the class **Product** should be implemented. It is a **base class** for any type of food and drink.

The class should receive **name: str** and **quantity: int** upon **initialization**.It should also have two methods:

* **decrease(quantity: int)** – decreases the quantity of the product only if there is enough
* **increase(quantity: int)** – increases the quantity of the product with the given one

In the file **drink.py** the class **Drink** should be implemented. The class should **inherit** from the **Product** class. An instance of the **Drink** class will have **name** and **quantity** of **10**.

In the **food.py** file the **Food** class should be implemented. The class should **inherit** from the **Product** class. An instance of the **Food** class will have **name** and **quantity** of **15**.

In the **product\_repository.py** file the class **ProductRepository** should be implemented. It is a **repository** for all the **products** that are delivered to the grocery shop.

The class should have **products:** **list** – **empty** list upon initialization that will contain **all products** (objects). Also, the class should have 3 instance methods:

* add(product: Product) - adds a product to the repository
* find(product\_name: str) - returns a product (object) with that name
* remove(product\_name) - removes a product from the repository

Override the \_\_repr\_\_ method, so it returns an information for all the products in the repository in the format:

"{product\_name1}: {quantity1}"

"{product\_name2}: {quantity2}"

...

"{product\_nameN}: {quantityN}"