Exercises: Inheritance

This document defines the exercises for the "Python OOP" course at @Software University.

Please submit your solutions (source code) to all the below-described problems in Judge.

1. Person

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

Create a **Child** class that inherits a **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.

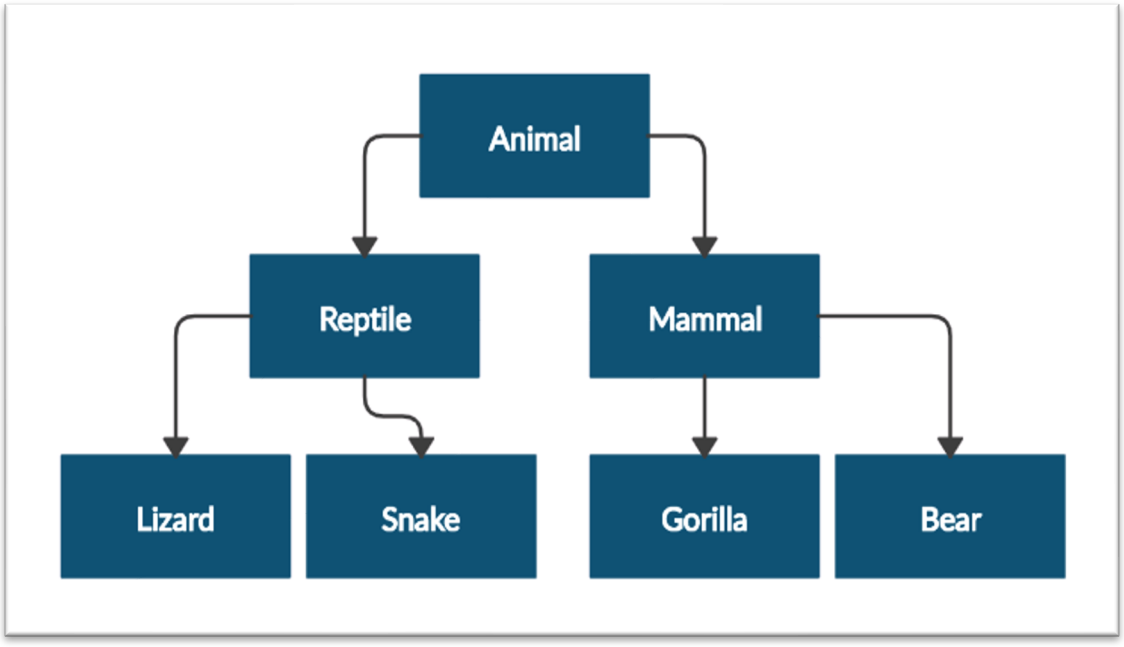
Examples

Test Code	Output

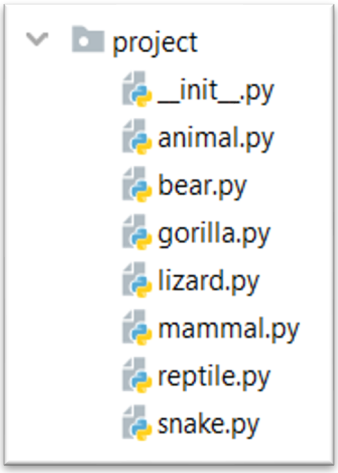
`person = Person("Peter", 25)`
`child = Child("Peter Junior", 5)`
`print(person.name)`
`print(person.age)`
`print(child.__class__.__bases__[0].__name__)`
	`Peter`
	`25`
	`Person`

2. Zoo

Create a **zoo** project that 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. **Except for the Animal class, each 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 a constructor, which accepts one parameter: **`name`**

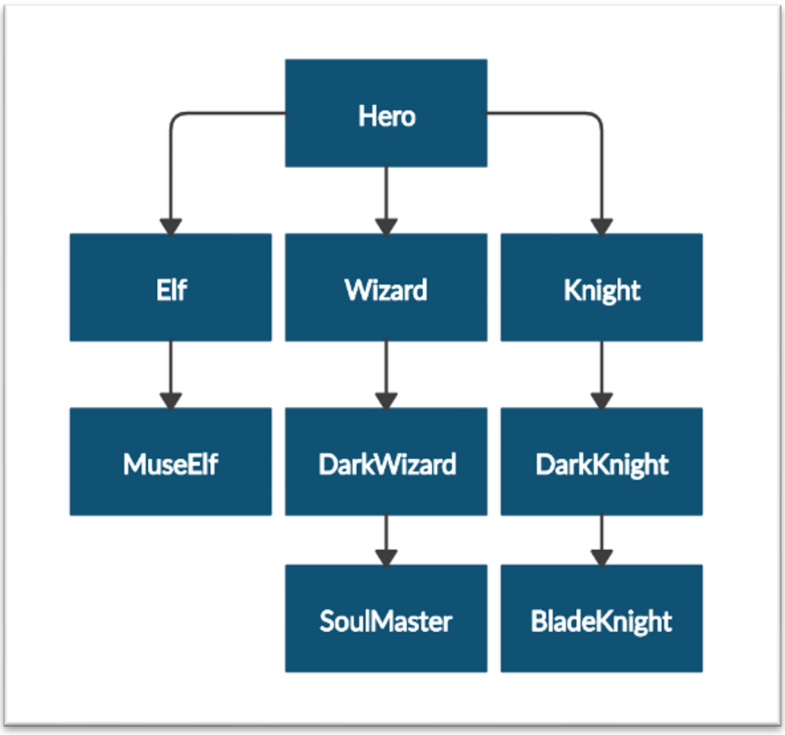
Examples

Test Code	Output

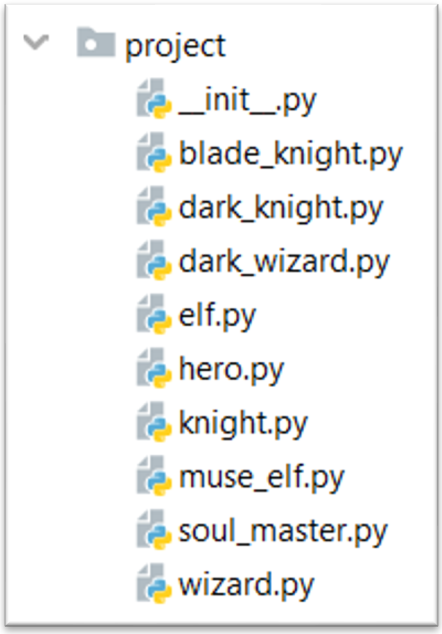
`mammal = Mammal("Stella")`
`print(mammal.__class__.__bases__[0].__name__)`
`print(mammal.name)`
`lizard = Lizard("John")`
`print(lizard.__class__.__bases__[0].__name__)`
`print(lizard.name)`
	`Animal`
	`Stella`
	`Reptile`
	`John`

3. 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}"`**

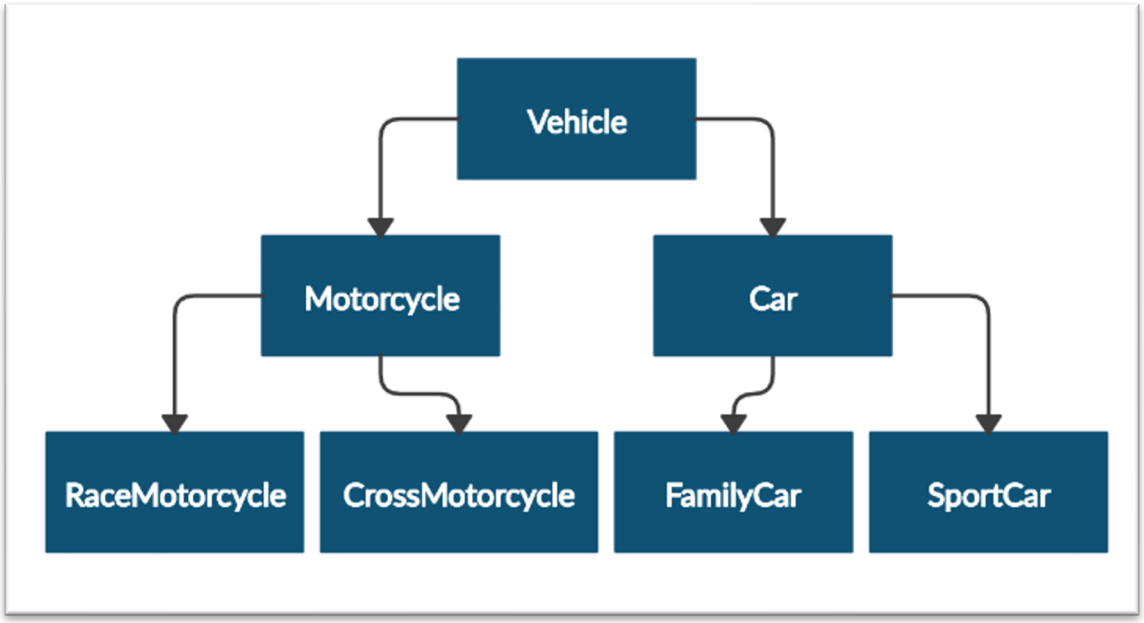
Examples

Test Code	Output

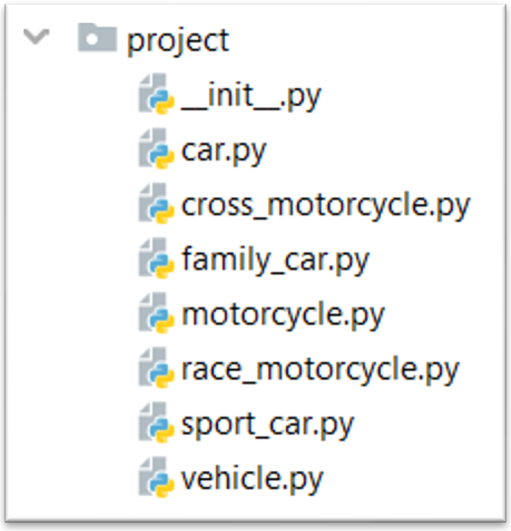
`hero = Hero("H", 4)`
`print(hero.username)`
`print(hero.level)`
`print(str(hero))`
`elf = Elf("E", 4)`
`print(str(elf))`
`print(elf.__class__.__bases__[0].__name__)`
`print(elf.username)`
`print(elf.level)`
	`H`
	`4`
	`H of type Hero has level 4`
	`E of type Elf has level 4`
	`Hero`
	`E`
	`4`

4. Need for Speed

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



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** - represents the fuel consumption per kilometer
- **`fuel`: float** - represents the quantity of fuel in a specific vehicle
- **`horse_power`: int**

Upon initialization, the class should receive **`fuel`** and **`horse_power`**. The **`DEFAULT_FUEL_CONSUMPTION`** value should be set to the **`fuel_consumption`** value.

Each class should have the following methods:

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

The default fuel consumption for the different vehicles is:

- **`Vehicle`** is **1.25**
- **`SportCar`** is **10**
- **`RaceMotorcycle`** is **8**
- **`Car`** is **3**

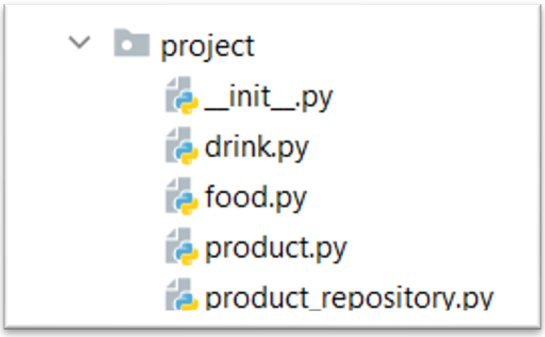
Examples

Test Code	Output

`vehicle = Vehicle(50, 150)`
`print(Vehicle.DEFAULT_FUEL_CONSUMPTION)`
`print(FamilyCar.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`
	`3`
	`50`
	`150`
	`1.25`
	`50`
	`0`
	`0`
	`Car`

5. Shop

Maria is expanding her business, and today, she is opening a grocery shop. You are hired to write a program that 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 3 additional methods:

- **`decrease(quantity: int)`** - decreases the quantity of the product only if there is enough
- **`increase(quantity: int)`** - increases the quantity of the product
- **`__repr__()`** - override the method, so it returns the name of the product

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 a **`name`** and a **`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 a **name** and a **quantity** of **15**.

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

The class should have **products: list** - an **empty** list, which will contain **all products** (objects). Also, the class should have **4 additional 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
- **__repr__()** - override the method, so it returns information for all products in the repository:

"{product_name1}: {quantity1}"

{product_name2}: {quantity2}

…

{product_nameN}: {quantityN}"

Examples

Test Code	Output
food = Food("apple") drink = Drink("water") repo = ProductRepository() repo.add(food) repo.add(drink) print(repo.products) print(repo.find("water")) repo.find("apple").decrease(5) print(repo)	[apple, water] water apple: 10 water: 10