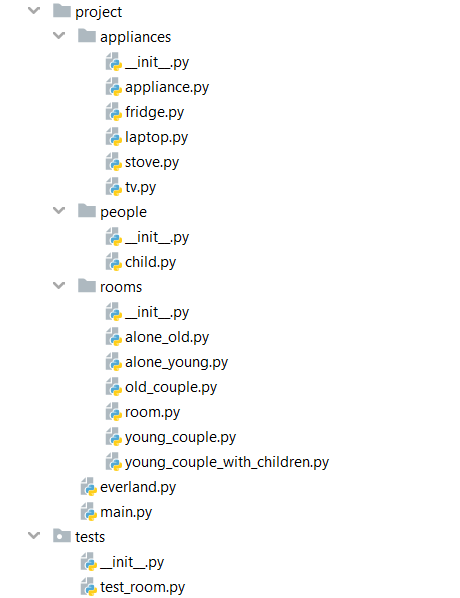
# Hotel Everland

*It is summertime, and you are hired to calculate the expenses and profit of the most popular hotel in town: Everland.*

You will be provided with a **skeleton** that includes all the folders and files that you will need.

***Note: You are not allowed to change the folder and file structure and change their names!***



# Judge Upload

For the **first 2 problems**, create a **zip** file with the name **project** and upload it to the judge system

For the **last problem**, create a **zip** file with the name **tests** and upload it to the judge system

# Structure (Problem 1) and Functionality (Problem 2)

Our first task is to implement the **structure and functionality** of all the classes (properties, methods, inheritance, etc.)

## Class Appliance

In the file **appliance.py**, the class **Appliance** should be implemented:

### Structure

The class should have the following attributes:

* **cost: float** - passed upon **initialization.** The cost is for a **single day**

### Methods

#### \_\_init\_\_(cost: float)

The **\_\_init\_\_** method should receive a **cost: float**

#### get\_monthly\_expense()

The method should **return** the cost for a **month (30 days)**

## Class Fridge

In the **fridge.py** file, the class **Fridge** should be implemented

### Structure

The class should **inherit** from the **Appliance** class

### Methods

#### \_\_init\_\_()

An instance of the **Fridge** class will have a **cost** of **1.2**

## Class Laptop

In the **laptop.py** file, the class **Laptop** should be implemented

### Structure

The class should **inherit** from the **Appliance** class

### Methods

#### \_\_init\_\_()

An instance of the **Laptop** class will have a **cost** of **1**

## Class Stove

In the **stove.py** file, the class **Stove** should be implemented

### Structure

The class should **inherit** from the **Appliance** class

### Methods

#### \_\_init\_\_()

An instance of the **Stove** class will have a **cost** of **0.7**

## Class TV

In the **tv.py** file, the class **TV** should be implemented

### Structure

The class should **inherit** from the **Appliance** class

### Methods

#### \_\_init\_\_()

An instance of the **TV** class will have a **cost** of **1.5**

## Class Child

In the **child.py** file, the class **Child** should be implemented

### Structure

The class should have the following attributes:

* **cost: float** – the money that the kid requires for a day

### Methods

#### \_\_init\_\_(food\_cost: int, \*toys\_cost)

**Sum** the **food\_cost** with the **cost of each toy** and set the **cost** attribute to the **result**

## Class Room

In the **room.py** file, the class **Room** should be implemented

### Structure

The class should have the following attributes:

* **family\_name: str** – **passed** upon initialization
* **budget: float** – **passed** upon initialization
* **members\_count: int** – **passed** upon initialization
* **children: list** – an **empty list** that will contain all kids in that room (objects)
* **expenses** – **0 by default**. If it is set to a negative number, raise **ValueError** with the message **"Expenses cannot be negative"**

### Methods

#### \_\_init\_\_(name: str, budget: float, members\_count: int)

Set the **attributes** to the **given values**

#### calculate\_expenses(\*args)

Each **element** of **args** will be a **list** (with **children** or **appliances**). Calculate the **total cost** for a month (**30 days**) of all elements in the lists and **set** the **expenses attribute** to the result.

## Class AloneOld

In the **alone\_old.py** file, the **AloneOld** class should be implemented

### Structure

The **AloneOld** class should inherit from the **Room** class.

### Attributes

Apart from the attributes of the **Room** class, it should have a **room\_cost** attribute equal to **10**

### Methods

#### \_\_init\_\_(family\_name: str, pension: float)

This room has only **one member**, and the **budget** equals the **pension** of the person.

## Class AloneYoung

In the **alone\_young.py** file, the **AloneYoung** class should be implemented

### Structure

The **AloneYoung** class should inherit from the **Room** class.

### Attributes

Apart from the attributes of the **Room** class, it should have a **room\_cost** attribute equal to **10** and a **list** of **appliances** (a **tv**)

### Methods

#### \_\_init\_\_(family\_name: str, salary: float)

This room has only **one member,** and the **budget** equals the **salary** of the person.

Calculate the **expenses** of each appliance.

## Class OldCouple

In the **old\_couple.py** file, the **OldCouple** class should be implemented

### Structure

The **OldCouple** class should inherit from the **Room** class.

### Attributes

Apart from the attributes of the Room class, the **OldCouple** class should have a **room\_cost** attribute equal to **15** and a **list** of **appliances** (a **tv**, a **fridge,** and a **stove** for **each person**)

### Methods

#### \_\_init\_\_(family\_name: str, pension\_one: float, pension\_two: float)

This room has **two members,** and the **budget** equals the **two pensions** of the people.

Calculate the **expenses** of each appliance.

## Class YoungCouple

In the **young\_couple.py** file, the **YoungCouple** class should be implemented

### Structure

The **YoungCouple** class should inherit from the **Room** class.

### Attributes

Apart from the attributes of the Room class, the **YoungCouple** class should have a **room\_cost** attribute equal to **20** and a **list** of **appliances** (a **tv**, a **fridge,** and a **laptop** for **each person**)

### Methods

#### \_\_init\_\_(family\_name: str, salary\_one: float, salary\_two: float)

This room has **two members,** and the **budget** equals the **two salaries** of the people.

Calculate the **expenses** of each appliance.

## Class YoungCoupleWithChildren

In the **young\_couple\_with\_children.py** file, the **YoungCoupleWithChildren** class should be implemented

### Structure

The **YoungCoupleWithChildren** class should inherit from the **Room** class.

### Attributes

Apart from the attributes of the Room class, the **YoungCoupleWithChildren** class should have a **room\_cost** attribute equal to **30**, a **list** of **children**, and a **list** of **appliances** (a **tv**, a **fridge**, and a **laptop** for **each person, including the children**)

### Methods

#### \_\_init\_\_(family\_name: str, salary\_one: float, salary\_two: float, \*children)

Add the **children** to the children attribute. Each **child** will be an **instance** of the **Child** class. This room can have a **different number** of **members** (parents + children), the **budget** equals to the **two salaries** of the people.

Calculate the **expenses** (appliances and children expenses).

## Class Everland

### Attributes

* **rooms: list** – an empty list that will contain all rooms (objects)

### Methods

#### \_\_init\_\_()

Set the **rooms** attribute to an **empty list**

#### add\_room(room: Room)

Add the **room** in the **rooms list**

#### get\_monthly\_consumptions()

Calculate the **expenses** of each room + the **room\_cost** and return the result in the following format: **"Monthly consumption: {total\_consumption}$."**

#### pay()

Each **room represents** one of the following **strings**:

* If the **budget** of the family **is enough** to pay for the month – **"{family\_name} paid {expenses+room\_cost}$ and have {new\_budget}$ left."** and **reduce the budget** of the family
* If the **budget** is **NOT** **enough** to pay for the month – **"{family\_name} does not have enough budget and must leave the hotel."** and **remove** the **room** from the rooms list

Return all the information by **joining** the strings by a **new line**

#### status()

Return **information** about the hotel. If there are **children** in the room, **print them first**, and then the **appliances monthly cost**. The result should be in the following format:

**Total population: {all\_people\_in\_the\_hotel}  
{room\_name} with {members} members. Budget: {current\_budget}$, Expenses: {expenses}$  
--- Child {n} monthly cost: {cost\_for\_one\_month}$  
… {rest of the children if any}  
--- Appliances monthly cost: {cost\_of\_all\_appliances\_for\_one\_month}$  
… {rest of the rooms if any}**

***Note: All the numbers must be formatted to the second digit***

# Examples

|  |
| --- |
| **Test Code** |
| from rooms.young\_couple import YoungCouple  from rooms.young\_couple\_with\_children import YoungCoupleWithChildren  from people.child import Child  from everland import Everland  everland = Everland()  def test\_one():  young\_couple = YoungCouple("Johnsons", 150, 205)  child1 = Child(5, 1, 2, 1)  child2 = Child(3, 2)  young\_couple\_with\_children = YoungCoupleWithChildren("Peterson", 600, 520, child1, child2)  everland.add\_room(young\_couple)  everland.add\_room(young\_couple\_with\_children)  print(everland.get\_monthly\_consumptions())  print(everland.pay())  print(everland.status())  if \_\_name\_\_ == "\_\_main\_\_":  test\_one() |
| **Output** |
| Monthly consumtions: 1136.00$.  Johnsons paid 242.00$ and have 113.00$ left.  Peterson paid 894.00$ and have 226.00$ left.  Total population: 6  Johnsons with 2 members. Budget: 113.00$, Expenses: 222.00$  --- Appliances monthly cost: 222.00$  Peterson with 4 members. Budget: 226.00$, Expenses: 864.00$  --- Child 1 monthly cost: 270.00$  --- Child 2 monthly cost: 150.00$  --- Appliances monthly cost: 444.00$ |
| **Comment** |
| Johnsons have expenses of 242$   * Room – 20$ * 2 Laptops – 60$ (30 \* 1 each) * 2 Fridges – 72$ (30 \* 1.2 each) * 2 TV's – 90$ (30 \* 1.5 each)   Petersons have expenses of 894$   * Room – 30$ * 4 Laptops – 120$ (30 \* 1 each) * 4 Fridges – 144$ (30 \* 1.2 each) * 4 TV's – 180$ (30 \* 1.5 each) * Child 1 – 270$ * Child 2 – 150$   Total consumption: 242 + 894 = 1136 |

# Problem 3. Unit Tests

You will **be provided with another skeleton** for this problem. You should **write tests** for the **SchoolReportCard** class. It will have some **properties**, **methods**, and a **constructor**, all of them **working properly**. You are **NOT ALLOWED** to change the class. Cover the whole class with unit tests to make sure that the class is working as intended. Submit **only the test** folder.