# Lab: Unit Testing

This document defines the exercises for the ["Python OOP" course at @Software University.](https://softuni.bg/trainings/4371/python-oop-february-2024)

Please submit your solutions (source code) to all the below-described problems in [Judge](https://judge.softuni.org/Contests/1948/Testing-Lab).

## Test Worker

Load the provided skeleton in the IDE you use. Add new project **Tests.**

**class** Worker:  
  
 **def** \_\_init\_\_(self, name, salary, energy):  
 self.name = name  
 self.salary = salary  
 self.energy = energy  
 self.money = 0  
  
 **def** work(self):  
 **if** self.energy <= 0:  
 **raise** Exception(**'Not enough energy.'**)  
  
 self.money += self.salary  
 self.energy -= 1  
  
 **def** rest(self):  
 self.energy += 1  
  
 **def** get\_info(self):  
 **return f'{**self.name**} has saved {**self.money**} money.'**

Create a class WorkerTests

\* In Judge, you must submit just the **WorkerTests** class, with the **unittest** **module imported** and the **main** **block**.

Create the following tests:

* Test if the worker is initialized with the correct name, salary, and energy
* Test if the worker's energy is incremented after the rest method is called
* Test if an error is raised if the worker tries to work with negative energy or equal to 0
* Test if the worker's money is increased by his salary correctly after the work method is called
* Test if the worker's energy is decreased after the work method is called
* Test if the **get\_info** method returns the proper string with the correct values

## Test Cat

**class** Cat:  
  
 **def** \_\_init\_\_(self, name):  
 self.name = name  
 self.fed = **False** self.sleepy = **False** self.size = 0  
  
 **def** eat(self):  
 **if** self.fed:  
 **raise** Exception(**'Already fed.'**)  
  
 self.fed = **True** self.sleepy = **True** self.size += 1  
  
 **def** sleep(self):  
 **if not** self.fed:  
 **raise** Exception(**'Cannot sleep while hungry'**)  
  
 self.sleepy = **False**

Create a class CatTests

\* In Judge, you must submit just the CatTests class, with the **unittest** **module imported** and the **main** **block**.

Create the following tests:

* The cat's size is increased after eating
* Cat is fed after eating
* Cat cannot eat if already fed, raises an error
* Cat cannot fall asleep if not fed, raises an error
* Cat is not sleepy after sleeping

### Hints

Follow the logic of the previous problem

## List

You are provided with a class **IntegerList**. It should **only store integers**. **The constructor should set the initial integers**. They are stored **as a list**. **IntegerList** has a functionality to **add**, **remove\_index**, **get, insert, get the biggest number, and get the index of an element**. Your task is to **test the class**.

***Note: You are not allowed to change the structure of the provided code***

### Constraints

* **add** operation, should **add an element** and return the list.
  + If the element is not an integer, a **ValueError** is thrown
* **remove\_index** operation removes the element on that index and returns it.
  + If the index is out of range, an **IndexError** is thrown
* **\_\_init\_\_** should only take integers, and store them
* **get** should return the specific element
  + If the index is out of range, an **IndexError** is thrown
* **insert**
  + If the index is out of range, **IndexError** is thrown
  + If the element is not an integer, **ValueError** is thrown
* **get\_biggest**
* **get\_index**

### Hint

Do not forget to **test the constructor**

## Car Manager

You are provided with a simple project **containing only one class** - **Car**. The provided class is simple - its **main point is to represent some of the functionality of a Car**. **Each car contains information** about its **make**, **model**, **fuel consumption**, **fuel amount**, and **fuel capacity**. Also, **each Car can add some fuel** to its tank by refueling and can travel a distance by **driving**. In order to be driven, our **Car** needs to **have enough fuel**. Everything in the provided skeleton is working perfectly fine, and **you mustn't change it**.

Your job now is to **write unit tests on the provided project** and **its functionality**. You should test **every part** of the code inside the **Car** class:

* You should test **the constructor**
* You should test **all the methods** and **validations inside the class**

### Constraints

* Everything in the provided skeleton is working perfectly fine
* You must not change anything in the project structure
* Any part of validation should be tested
* There is no limit on the tests you can write but keep your attention on the main functionality

***Note: You are not allowed to change the structure of the provided code***

*"Brum…Brum…Brum-suuuututututu…"*