Exercise: SOLID

Problems for exercise and homework for the Python OOP Course @SoftUni.

1. Workers

You are provided with a code on which you have to apply the **DIP** (Dependency Inversion Principle) so that when adding new **worker classes**, the Manager class will work properly.

Examples

Before	Result

`worker = Worker()`
`manager = Manager()`
`manager.set_worker(worker)`
`manager.manage()`
 	
`super_worker = SuperWorker()`
`try:`
`    manager.set_worker(super_worker)`
`except AssertionError:`
`    print("manager fails to support super_worker....")`
	`I'm working!!`
	`manager fails to support super_worker....`

After	**Result**

`worker = Worker()`
`manager = Manager()`
`manager.set_worker(worker)`
`manager.manage()`
 	
`super_worker = SuperWorker()`
`try:`
`    manager.set_worker(super_worker)`
`    manager.manage()`
`except AssertionError:`
`    print("manager fails to support super_worker....")`
	`I'm working!!`
	`I work very hard!!!`

2. Workers - Updated

You are provided with a code on which you have to apply the **ISP** (Interface Segregation Principle) by **splitting** the **Worker** class into two classes (**Workable** and **Eatable**), so the **Robot** class no longer needs to implement the **eat** method

Examples

Before	Result

`manager = Manager()`
`manager.set_worker(Worker())`
`manager.manage()`
	`I'm normal worker. I'm working.`
	`Lunch break....(5 secs)`
	`I'm super worker. I work very hard!`

manager.lunch_break() manager.set_worker(SuperWorker()) manager.manage() manager.lunch_break() manager.set_worker(Robot()) manager.manage() manager.lunch_break()	Lunch break....(3 secs) I'm a robot. I'm working.... I don't need to eat....
After	**Result**

work_manager = WorkManager()
break_manager = BreakManager()
work_manager.set_worker(Worker())
break_manager.set_worker(Worker())
work_manager.manage()
break_manager.lunch_break()
 	
work_manager.set_worker(SuperWorker())
break_manager.set_worker(SuperWorker())
work_manager.manage()
break_manager.lunch_break()
 	
work_manager.set_worker(Robot())
work_manager.manage()
try:
    break_manager.set_worker(Robot())
    break_manager.lunch_break()
except:
    pass
	I'm normal worker. I'm working.
	Lunch break....(5 secs)
	I'm super worker. I work very hard!
	Lunch break....(3 secs)
	I'm a robot. I'm working....

3. Prisoner

You are provided with a code containing a class **Prisoner** and a **class Person**. A **prisoner** is obviously a **person**, but since a **prisoner** is **not free** to move an arbitrary distance, the **Person** class can be named **FreePerson**, then the idea that a **Prisoner inherits FreePerson** is **wrong**. Rewrite the code and apply the **LSP** (Liskov Substitution Principle).

Examples

Before	Result
prisoner = Prisoner() print("The prisoner trying to walk to north by 10 and east by -3.") try:	The prisoner trying to walk to north by 10 and east by -3. The location of the prison: [3, 3] The current position of the prisoner: [0, 13]

prisoner.walk_north(10) prisoner.walk_east(-3) except: pass print(f"The location of the prison: {prisoner.PRISON_LOCATION}") print(f"The current position of the prisoner: {prisoner.position}")	
After	**Result**
prisoner = Prisoner() print("The prisoner trying to walk to north by 10 and east by -3.") try: prisoner.walk_north(10) prisoner.walk_east(-3) except: pass print(f"The location of the prison: {prisoner.PRISON_LOCATION}") print(f"The current position of the prisoner: {prisoner.position}")	The prisoner trying to walk to north by 10 and east by -3. The location of the prison: (3, 3) The current position of the prisoner: (3, 3)

4. Shapes

You are provided with code containing **class `Rectangle`** and **class `AreaCalculator`**. Refactor the code using the **Open/Closed Principle** so that the code is open for extension (adding **more shapes**) but closed for modification.

Examples

Before	Result
shapes = [Rectangle(2, 3), Rectangle(1, 6)] calculator = AreaCalculator(shapes) print("The total area is: ", calculator.total_area)	The total area is: 12
After	**Result**
shapes = [Rectangle(1, 6), Triangle(2, 3)] calculator = AreaCalculator(shapes) print("The total area is: ", calculator.total_area)	The total area is: 9.0

5. Emails

You are provided with code containing **class `IEmail`** and **class `Email`**. The code does not follow the principle of **single responsibility** (the Email class has **2 responsibilities**). Create a new **class - `IContent`**, and a class that inherits it called **`MyContent`** to split the responsibilities.

Examples

Before	Result
email = Email('IM', 'MyML') email.set_sender('qmal') email.set_receiver('james') email.set_content('Hello, there!') print(email)	Sender: I'm qmal Receiver: I'm james Content: <myML> Hello, there! </myML>
After	**Result**
email = Email('IM') email.set_sender('qmal') email.set_receiver('james') content = MyContent('Hello, there!') email.set_content(content) print(email)	Sender: I'm qmal Receiver: I'm james Content: <MyML>Hello, there!</MyML>