Lab: Encapsulation

Problems for in-class lab for the Python OOP Course @SoftUni.
Submit your solutions in the SoftUni judge system at https://judge.softuni.org/Contests/1938.

1. Person

Create a class called **Person**. Upon initialization, it should receive a **name** and an **age**. Name mangle **the name** and **the age** attributes (should not be accessed outside the class). Create two **instance methods** called **get_name** and **get_age** to return the values of the private attributes.

Examples

Test Code	Output

`person = Person("George", 32)`
`print(person.get_name())`
`print(person.get_age())`
	`George`
	`32`

2. Mammal

Create a class called **Mammal**. Upon initialization, it should receive a **name**, a **type**, and a **sound**. Create a **class attribute** called **kingdom** which should not be accessed outside the class and set it to be **"animals"**. Create **three more instance methods**:

- **make_sound()** - returns a string in the format **"{name} makes {sound}"**
- **get_kingdom()** - returns the private kingdom attribute
- **info()** - returns a string in the format **"{name} is of type {type}"**

Examples

Test Code	Output

`mammal = Mammal("Dog", "Domestic", "Bark")`
`print(mammal.make_sound())`
`print(mammal.get_kingdom())`
`print(mammal.info())`
	`Dog makes Bark`
	`animals`
	`Dog is of type Domestic`

3. Profile

Create a class called **Profile**. Upon initialization, it should receive:

- **username: str** - the username should be between 5 and 15 characters (inclusive). If it is not, raise a **ValueError** with the message **"The username must be between 5 and 15 characters."**
- **password: str** - the password must be at **least 8 characters** long; it must contain at least **one upper case letter** and at least **one digit**. If it does not, raise a **ValueError** with the message **"The password must be 8 or more characters with at least 1 digit and 1 uppercase letter."**

Hint: Use **Getters** and **Setters** to name-mangle them.

Override the **__str__()** method of the base class, so it returns: **"You have a profile with username: "{username}" and password: {"*" with the length of password}"**.

Examples

Test Code	Output
`profile_with_invalid_password = Profile('My_username', 'My-password')`	`ValueError: The password must be 8 or more characters with at least 1 digit and 1 uppercase letter.`
`profile_with_invalid_username = Profile('Too_long_username', 'Any')`	`ValueError: The username must be between 5 and 15 characters.`
`correct_profile = Profile("Username", "Passw0rd")` `print(correct_profile)`	`You have a profile with username: "Username" and password: ********`

4. Email Validator

Create a class called **`EmailValidator`**. Upon initialization it should receive:

- **`min_length`** (of the username; example: in **"peter@gmail.com"** **"peter"** is the **username**)
- **`mails`** (**list** of the **valid mails**; example: **"gmail"**, **"abv"**)
- **`domains`** (**list** of **valid domains**; example: **"com"**, **"net"**)

Create **three methods that should not be accessed outside** the class:

- **`is_name_valid(name)`** - returns whether the name is **greater than or equal to the `min_length`** (True/False)
- **`is_mail_valid(mail)`** - returns whether the **mail is in the possible mails list** (True/False)
- **`is_domain_valid(domain)`** - returns whether the **domain is in the possible domains list** (True/False)

Create one **public method**:

- **`validate(email)`** - using the **three methods** returns whether the **email is valid** (True/False)

Examples

Test Code	Output

`mails = ["gmail", "softuni"]`
`domains = ["com", "bg"]`
`email_validator = EmailValidator(6, mails, domains)`
`print(email_validator.validate("pe77er@gmail.com"))`
`print(email_validator.validate("georgios@gmail.net"))`
`print(email_validator.validate("stamatito@abv.net"))`
`print(email_validator.validate("abv@softuni.bg"))`
	`True`
	`False`
	`False`
	`False`

5. Account

Create a class called **`Account`**. Upon initialization, it should receive an **`id`**, a **`balance`**, and a **`pin`** (all numbers). The **`pin`** and the **`id`** should be **private instance attributes**, and the **`balance`** should be a **public attribute**. Create **two public instance methods**:

- **`get_id(pin)`** - if the given **`pin`** is correct, return the **`id`**, otherwise, return **`"Wrong pin"`**
- **`change_pin(old_pin, new_pin)`** - if the old pin is **correct**, **change** it to the new one and return **`"Pin changed"`**, otherwise return **`"Wrong pin"`**

Examples

Test Code	Output

`account = Account(8827312, 100, 3421)`
`print(account.get_id(1111))`
`print(account.get_id(3421))`
`print(account.balance)`
`print(account.change_pin(2212, 4321))`
`print(account.change_pin(3421, 1234))`
	`Wrong pin`
	`8827312`
	`100`
	`Wrong pin`
	`Pin changed`