

Міністерство освіти і науки України
Національний університет «Львівська політехніка»
Кафедра «Електронних обчислювальних машин»



Звіт
з лабораторної роботи № 9
з дисципліни: «Кросплатформенні засоби програмування»
на тему: «Основи об'єктно - орієнтованого програмування у Python»

Виконав:

студент групи КІ-306

Щирба Д.В.

Перевірив:

доцент кафедри ЕОМ

Іванов Ю. С.

Мета роботи: оволодіти навиками реалізації парадигм об'єктно-орієнтованого програмування використовуючи засоби мови Python.

Завдання (варіант № 28) – похідний клас Енергозберігаюча лампочка

1. Написати та налагодити програму на мові Python згідно варіанту. Програма має задовольняти наступним вимогам:

- класи програми мають розміщуватися в окремих модулях в одному пакеті;
- точка входу в програму (main) має бути в окремому модулі;
- мають бути реалізовані базовий і похідний класи предметної області згідно варіанту;
- програма має містити коментарі.

2. Завантажити код на GitHub згідно методичних вказівок по роботі з GitHub.

3. Скласти звіт про виконану роботу з приведенням тексту програми, результату її виконання та фрагменту згенерованої документації та завантажити його у ВНС.

Вихідний код програми:

Файл Lightbulb.py

```
class Lightbulb:
    def __init__(self, model="Voltic", energy_consumption=10, color="white"):
        self.model = model
        self.energy_consumption = energy_consumption
        self.isOn = False
        self.color = color
        self.logFile = open("Lightbulb_Log.txt", "w")

    def turn_on(self):
        self.isOn = True
        self.logFile.write("Lightbulb turned on.\n")
        self.logFile.flush()

    def turn_off(self):
        self.isOn = False
        self.logFile.write("Lightbulb turned off.\n")
        self.logFile.flush()

    def change_color(self, new_color):
        self.color = new_color
        self.logFile.write(f"Color changed to {new_color}\n")
        self.logFile.flush()

    def change_model(self, new_model):
        self.model = new_model
        self.logFile.write(f"Model changed to {new_model}\n")
        self.logFile.flush()

    def get_model(self):
```

```

        return self.model

    def get_energy_consumption(self):
        return self.energy_consumption

    def change_energy_consumption(self, new_energy_consumption):
        self.energy_consumption = new_energy_consumption
        self.logFile.write(f"Energy consumption changed to {new_energy_consumption}
watts\n")
        self.logFile.flush()

    def luminous_efficiency(self):
        efficiency = 10 * self.energy_consumption
        self.logFile.write(f"Luminous efficiency of the lightbulb: {efficiency}
lumens/watt\n")
        self.logFile.flush()
        return efficiency

    def get_info(self):
        print(f"Model: {self.model}")
        print(f"Power (watts): {self.energy_consumption}")
        print(f"State: {'on' if self.isOn else 'off'}")
        print(f"Light color: {self.color}")

    def close_log_file(self):
        self.logFile.close()

```

Файл ES_Lightbulb.py

```

from Lightbulb import Lightbulb # Assuming Lightbulb class is in a separate module

class ES_Lightbulb(Lightbulb):
    def __init__(self, model="Voltic", energy_consumption=10, color="white",
brightness_level=5, daylight_sensor=False):
        super().__init__(model, energy_consumption, color)
        self.brightness_level = brightness_level
        self.daylight_sensor = daylight_sensor

    def get_brightness_level(self):
        return self.brightness_level

    def set_brightness_level(self, brightness_level):
        self.brightness_level = brightness_level
        self.logFile.write(f"Brightness level changed to {brightness_level}\n")
        self.logFile.flush()

    def increase_brightness(self, amount):
        if self.brightness_level + amount <= 10:
            self.brightness_level += amount
            self.logFile.write(f"Brightness level increased by {amount}\n")
            self.logFile.flush()

    def decrease_brightness(self, amount):
        if self.brightness_level - amount >= 1:
            self.brightness_level -= amount
            self.logFile.write(f"Brightness level decreased by {amount}\n")
            self.logFile.flush()

    def has_daylight_sensor(self):
        return self.daylight_sensor

    def toggle_daylight_sensor(self):
        self.daylight_sensor = not self.daylight_sensor
        state = "enabled" if self.daylight_sensor else "disabled"
        self.logFile.write(f"Daylight sensor {state}\n")

```

```

        self.logFile.flush()

    def auto_turn_on_at_brightness_threshold(self, threshold):
        if self.brightness_level >= threshold and not self.isOn:
            self.turn_on()

    def get_info(self):
        super().get_info()
        print(f"Brightness level: {self.brightness_level}")
        print(f"Daylight sensor: {'present' if self.daylight_sensor else 'absent'}")

```

Файл Lab9_Python.py

```

from ES_Lightbulb import ES_Lightbulb # Assuming ES_Lightbulb class is in a separate
module

def main():
    try:
        E77 = ES_Lightbulb()
        E77.get_brightness_level()
        E77.set_brightness_level(8)
        E77.increase_brightness(2)
        E77.decrease_brightness(1)

        E77.has_daylight_sensor()
        E77.toggle_daylight_sensor()

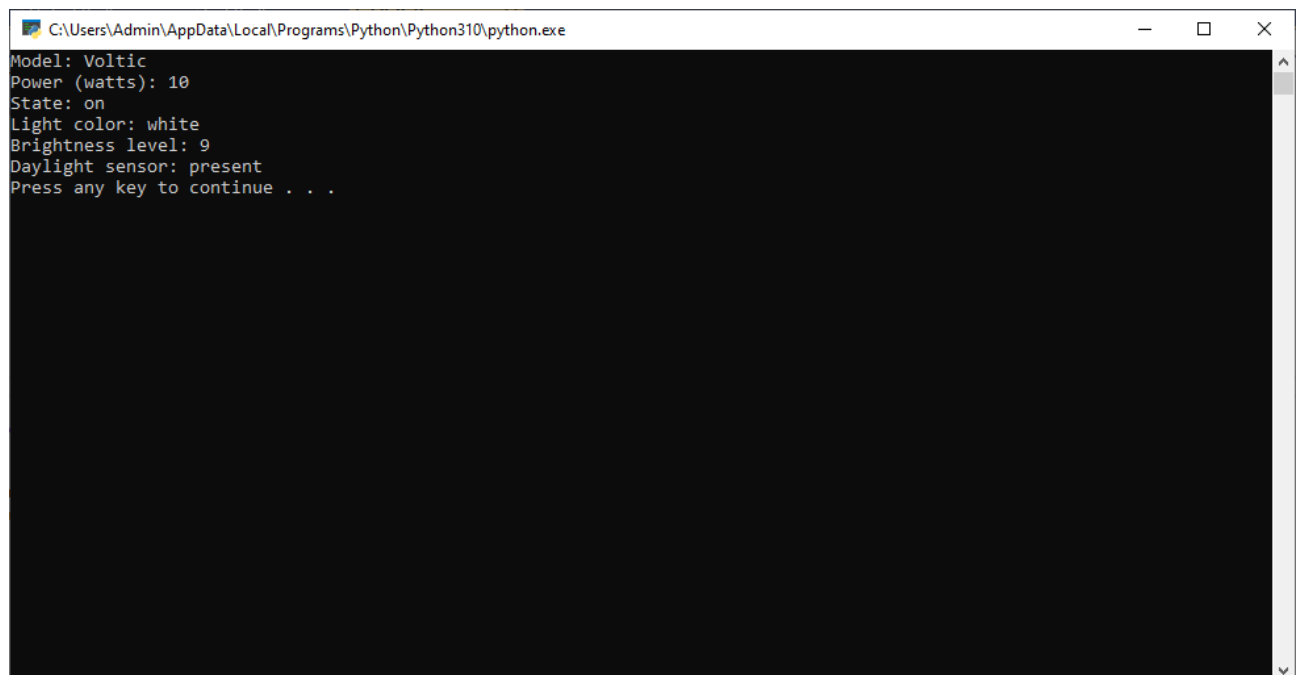
        E77.auto_turn_on_at_brightness_threshold(7)

        E77.get_info()
        E77.close_log_file()
    except FileNotFoundError as e:
        print(f"Error: {e}")

if __name__ == "__main__":
    main()

```

Результат виконання програми:



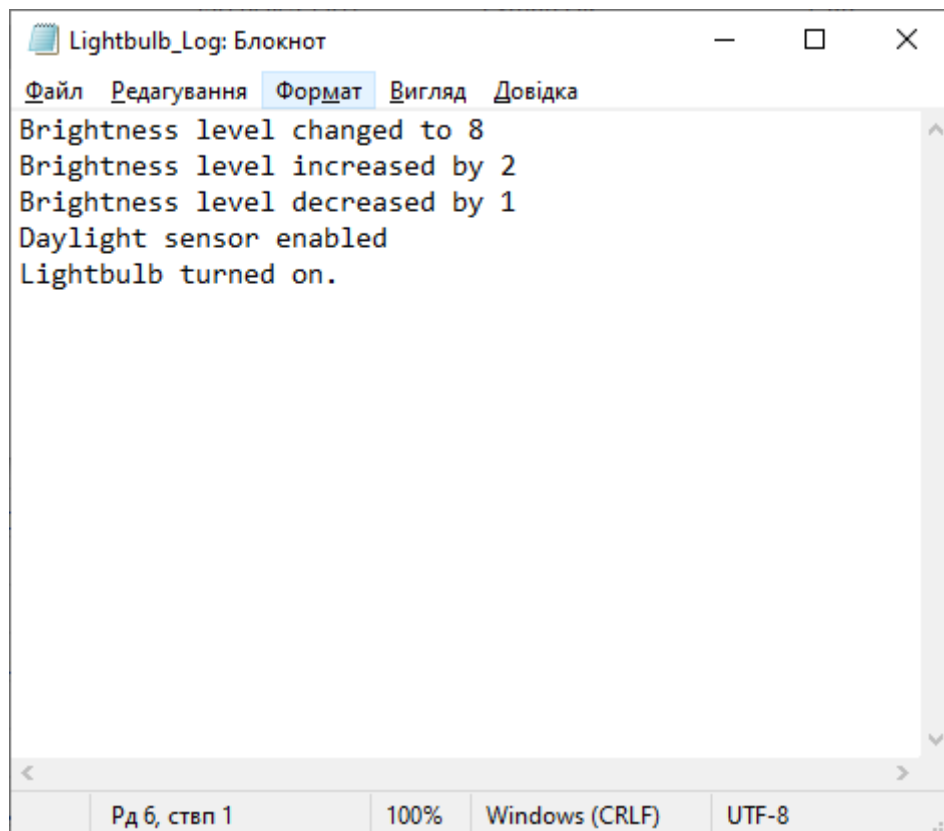
The screenshot shows a terminal window titled "C:\Users\Admin\AppData\Local\Programs\Python\Python310\python.exe". The output of the program is as follows:

```

Model: Voltic
Power (watts): 10
State: on
Light color: white
Brightness level: 9
Daylight sensor: present
Press any key to continue . . .

```

Результат виконання програми у файлі Lightbulb_Log.txt:



Відповіді на контрольні запитання:

1. **Модулі** - це файли в Python, які містять пайтонівський код. Модулі дозволяють організувати код в логічні блоки та використовувати його в інших програмах.
2. **Імпортувати модуль** можна за допомогою ключового слова **import**. Наприклад, **import math** імпортує модуль **math**.
3. **Оголошення класу** починається з ключового слова **class**, за яким слідує ім'я класу і двокрапка. Наприклад, **class MyClass:**.
4. У **класі** можуть міститися:
 - **Атрибути:** Змінні, які присвоюються об'єктам класу.
 - **Методи:** Функції, що визначають поведінку об'єктів класу.
 - **Конструктор:** Спеціальний метод **__init__**, який викликається при створенні нового об'єкта.
 - **Інші спеціальні методи:** Наприклад, **__str__** для представлення об'єкта у вигляді рядка.
5. **Конструктор класу** має ім'я **__init__** і викликається автоматично при створенні нового об'єкта. В ньому встановлюються початкові значення атрибутів.
6. **Спадкування** в Python означає отримання властивостей та методів від батьківського класу. Це реалізується шляхом вказання батьківського класу у визначенні дочірнього класу.
7. **Види спадкування:**
 - **Одиночне спадкування:** Клас успадковує властивості лише від одного батьківського класу.
 - **Множинне спадкування:** Клас успадковує властивості від багатьох батьківських класів.
8. **Небезпеки при множинному спадкуванні:**
 - **Конфлікти імен:** Можуть виникнути колізії імен методів або атрибутів між батьківськими класами.
 - **Складність розуміння та утримання:** Множинне спадкування може зробити код складнішим для розуміння та утримання.
9. **Класи-домішки (Mixin classes)** - це спеціальні класи, які призначені для розширення функціональності інших класів шляхом надання додаткових методів та атрибутів.

10. Функція `super()` при спадкуванні використовується для виклику методів батьківського класу в дочірньому класі. Вона дозволяє уникнути проблем з однойменними методами в дочірньому та батьківському класах.

Висновок:

Під час лабораторної роботи, я оволодів навиками реалізації парадигм об'єктно-орієнтованого програмування використовуючи засоби мови Python.