Міністерство освіти і наук України

Національний університет “Львівська політехніка”

Кафедра ЕОМ

ОСНОВИ ОБ’ЄКТНО-ОРІЄНТОВАНОГО ПРОГРАМУВАННЯ У PYTHON

Методичні вказівки
до лабораторної роботи № 9 з курсу “Кросплатформні засоби програмування”
для студентів базового напряму 6.123 - “Комп’ютерна інженерія”

Затверджено
на засіданні кафедри
”Електронні обчислювальні машини”
Протокол № 1 від 28.08.2023 р.

Львів – 2023

Основи об’єктно-орієнтованого програмування у Python: Методичні вказівки до лабораторної роботи № 9 з курсу “Кросплатформні засоби програмування” для студентів базового напряму 6.123 - “Комп’ютерна інженерія” / Укладач: Олексів М.В. – Львів: Національний університет “Львівська політехніка”, 2023, 19 с.

Укладач Олексів М. В., к.т.н.

Рецензент

Відповідальний за випуск: Мельник А. О., професор, завідувач кафедри

ОСНОВИ ОБ'ЄКТНО-ОРІЄНТОВАНОГО ПРОГРАМУВАННЯ У PYTHON

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

ТЕОРЕТИЧНІ ВІДОМОСТІ

Модулі

Модулем у Python називається файл з розширенням *.py. Ці файли можуть містити звичайні скрипти, змінні, функції, класи і їх комбінації. Python дозволяє структурувати код програм у різні модулі та доступатися до класів, функцій і змінних, які у них знаходяться з інших модулів. Для цього використовуються два оператори – `import` та `from-import`.

Оператор `import`

Оператор `import` дозволяє імпортувати модуль повністю, та доступатися до нього через назву модуля. Вона може бути вказана у будь-якому місці програми перед звертанням до елементів, які у ній містяться, але зазвичай її вказують на початку модуля. Для звертання до елементів модуля треба вказати назву модуля і після крапки вказати до якого елементу ви хочете звернутися.

Синтаксис

```
import назва_модуля
назва_модуля.елемент_модуля
```

Приклад

Файл my_print_module.py

```
def hello_print():
     print("Hello")
```

Файл printer.py

```
import my_print_module
my_print_module.hello_print ()
```

Для зручності оригінальну назву модуля можна змінити на псевдонім і доступатися до елементів модуля за допомогою псевдоніма.

Синтаксис

```
import оригінальна_назва_модуля as псевдонім
```

Приклад

```
import my_print_module as my_printer
```

Класи

Клас оголошується за допомогою ключового слова `class` після якого йде назва класу. Клас може містити:

- дані, які належать класу (статичні дані-члени класу);
- дані, які належать об'єкту класу;
- методи, які належать класу (статична методи);
- методи, які належать об'єкту класу.

Члени класу є лише публічні, проте Python забезпечує механізми, які дозволяють організувуати області видимості близькі за своєю суттю до protected і private. Це робиться шляхом використання нижнього підкреслення у назві членів класу. Одинарне нижнє підкреслення перед назвою члену класу робить за своїми властивостями схожим на захищений член класу, а подвійне – схожим на приватний член класу. Всі члени класу, що йому належать мають відступ у розмірі одного табулятора, або 4-ох пробілів від початку оголошення класу.

Статичні члени-дані класу оголошуються в класі як назва змінної і її початкове значення.

Дані, які належать об'єкту класу оголошуються в конструкторі з використанням ключового слова `self`, яке є посиланням на об'єкт класу:

```
self.<назва_змінної> = <початкове значення>
```

Статичні методи класу оголошуються в класі за правилами оголошення функцій.
Не статичні методи оголошуються в класі за правилами оголошення функцій, перший параметр якої є обов'язково `self`:

```
def <назва_методу>(self, <параметри>):
     тіло методу
```

Роль конструктора відіграє метод `__init__(self, <параметри>)`.

Доступ до статичних членів класу відбувається за допомогою назви класу:

```
<назва класу>.<назва статичного члену класу>.
```

Доступ до не статичних членів класу відбувається за допомогою назви об'єкту:

```
<назва об'єкту>.<назва не статичного члену класу>.
```

Параметри `self` передавати у метод при виклику не потрібно. Він передається неявно як і у інших мовах програмування.

Якщо клас не містить ніяких членів, то він має мати замість них ключову слово `pass`.

```
class <назва класу>:
    pass
```

Видалення окпремих членів класу або об’єкту загалом здійснюється за допомогою оператора `del`:

```
del <назва_об’єкту.назва_члену_даних> # видалення властивості

del <назва_об’єкту> # видалення об’єкту
```

Приклад:

```
class Pet:
    number = 0 # статична публічна змінна-член класу
    _average_height = 0 # статична захищена змінна-член класу
    def __init__(self, name, age, height):
        self.__name = name # не статична приватна змінна-член класу
        self.__age = age
        self.__height = height
        self._height_in_inch = height * 2.54 # не статична захищена змінна-член класу
        Pet._average_height = (Pet._average_height + height) / 2
        Pet.number = Pet.number + 1

    def __private_member(self, ...):
        """ приватний метод """
        <тіло>

    def _protected_member(self, ...):
        """ захищений метод """
        <тіло>

    def public_member(self, ...):
        """ загальнодоступний метод """
        <тіло>


    @staticmethod
    def public_static_member(...):
        """ загальнодоступний статичний метод """
        <тіло>
```

```
my_pet = Pet("Fluffy", 3, 15) # створення об’єкту my_pet
print(Pet.number) # звернення до статичних членів-даних
my_pet.public_member() # виклик не статичного методу
Pet.public_static_member() # виклик статичного методу
del my_pet # видалення об’єкту
```

Спадкування

Одинарне спадкування

Спадкування призначене для розширення функціональності існуючих класів шляхом утворення нових класів на базі вже існуючих. У Python усі класи спадкуються неявно від класу `object`. Python дозволяє реалізовувати як одинарне так і множинне спадкування. Для реалізації спадкування класи, які слід успадкувати вказуються у круглих дужках через кому після назви класу, який оголошується:

```
class <назва_класу> (<базовий_клас_1>, <базовий_клас_2>, ...):
    <тіло класу>
```

При одинарному спадкуванні похідний клас спадкує один базовий клас. Наприклад:

```
class Animal:
     def __init__(self, breed, age, height):
        self.__breed = breed # приватна змінна-член класу
        self.__age = age
        self.__height = height

     def get_breed(self):
        return self.__breed

     def get_age(self):
        return self.__age

     def get_height(self):
        return self.__heihgt

class Dog(Animal):
     def __init__(self, breed, age, height, voice):
        super().__init__(breed, age, height)
        self.__voice = voice

     def voice(self):
        print(self.__voice)
```

Явне звернення до членів будь-якого базового класу в межах ієрархії спадкування виконуєтсья за допомогою назви базового класу та його :

```
<назва_базового_класу>.<член_класу>.
```

Для виклику конструктора базового класу у тілі конструктора похідного класу слід викликати функцію `super()`, який повертає проксі об'єкт базового класу, та явно викликати з-під нього конструктор базового класу. Також функцію `super()` слід використовувати для звертання до інших членів базового класу.

Слід мати на увазі, що конструктор базового класу автоматично НЕ викликається з-під конструктора похідного класу як у більшості інших об'єктно-орієнтованих мов програмування. Його треба викликати самостійно, додавши виклик `super().__init__()` у метод `__init__()` похідного класу.

Множинне спадкування

Особливістю множинного спадкування є наявність кількох класів від яких одночасно відбувається спадкування. Тут можливі 2 випадки: Класи, що спадкуються від незалежних між собою класів та множинне спадкування від залежних між собою класів.

Множинне спадкування від незалежних між собою класів

При множинному спадкуванню від незалежних між собою класів усі базові класи по кожній гілці спадкування мають унікальних предків.

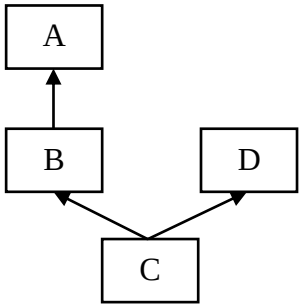

Рис. 1. Варіант множинного спадкування від незалежних між собою класів.

У звязку з тим, що конструктор (метод `__init()__`) у Python не викликає автоматично конструкторів базового класу, то їх слід викликати самостійно.

Якщо базовий клас не містить полів, які слід ініціалізовувати, то виклик конструктора цього базового класу можна пропустити, проте це не рекомендується робити.

Якщо методи `__init()__` різних класів приймають різні параметри, то:

1. Конструктори усіх класів необхідно реалізувати таким чином, щоб вони додатково приймали параметром словник параметрів у форматі ключ-значення (`**kwargs`). У цей словник попадатимуть усі параметри, які передані конструктору, але не згадані явно перед `**kwargs`.

2. Викликати конструктори базових класів для усіх класів в ієрархії. Це необхідно для коректної роботи механізму MRO (розглядається далі).

Приклад:

```
class Animal:
    def __init__(self, breed, age, height, **kwargs):
        super().__init__(**kwargs) # обов’язково має бути,
оскільки клас object спадкується 2 рази у ієрархії.
        self.__breed = breed
        self.__age = age
        self.__height = height
        print(f"Animal breed={self.__breed}, age={self.__age},
height={height}")

    def get_breed(self):
        return self.__breed

    def get_age(self):
        return self.__age

    def get_height(self):
        return self.__heihgt


class PetInfo:
    def __init__(self, address, vaccination, **kwargs):
        super().__init__(**kwargs) # обов’язково має бути,
оскільки клас object спадкується 2 рази у ієрархії.
        self.__address = address
        self.__vaccination = vaccination
        print(f"PetInfo address={self.__address},
vaccination={self.__vaccination}")

    def get_address(self):
        return self.__address

    def get_vaccination(self):
        return self.__vaccination

class DogPet(Animal, PetInfo):
    def __init__(self, breed, age, height, voice, address,
vaccination):
        super().__init__(breed=breed, age=age, height=height,
address=address, vaccination=vaccination)
```

```
        self.__voice = voice
        print(f"DogPet voice={self.__voice}")

    def voice(self):
        print(self.__voice)

x=DogPet("breed", 15, 10, "Bow", "address", True)
```

Вивід на екран:

```
PetInfo address=address, vaccination=True
Animal breed=breed, age=15, height=10
DogPet voice=Bow
```

Множинне спадкування від залежних між собою класів

При множинному спадкуванню від залежних між собою класів принаймі два базові класи по різних гілках спадкування мають принаймі одного спільного предка.

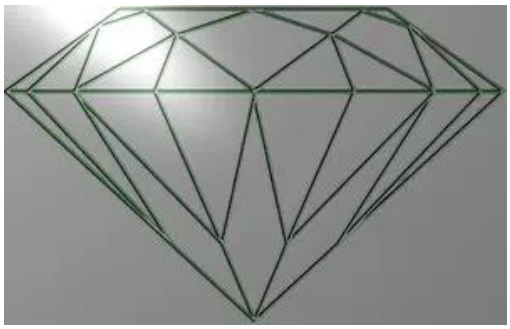

Рис. 2. «Діамант смерті»

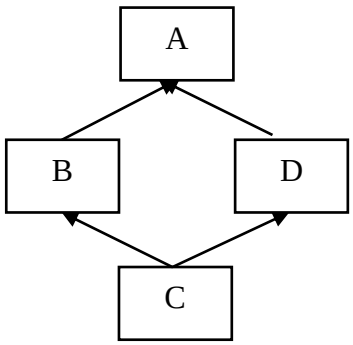

Рис. 3. Варіант множинного спадкування від залежних між собою класів.

Для коректної побудови усіх об’єктів при даному виді множинного спадкування для усіх класів слід задати метод `__init__()`, в якому здійснити виклик контруктора базового класу, викликавши `super().__init__()`.

Приклад:

```
class A:
     def __init__(self):
          super().__init__() # Можна опустити оскільки клас
object спадкується 1 раз у ієрархії через цей клас
          print("__init__() of A called")

class B(A):
     def __init__(self):
          super().__init__()
          print("__init__() of B called")

class C(A):
     def __init__(self):
          super().__init__()
          print("__init__() of C called")


     def foo(self):
          print("foo() of C called")


class D(B,C):
     def __init__(self):
          super().__init__()
          print("__init__() of D called")

x=D()
```

Вивід на екран:

```
__init__() of A called
__init__() of C called
__init__() of B called
__init__() of D called
```

Порядок виведення методів

Порядок виведення методів (Method Resolution Order - MRO) – це спосіб за допомогою якого деревовидна структура ієрархії спадкування класів представляється лінійно у вигляді списку. Він дозволяє Python з'ясувати, з якого предка потрібно викликати метод, якщо він не виявлений безпосередньо в класі-нащадку. Якщо в кожного нащадка є лише один предок, то завдання тривіальне. Відбувається висхідний пошук у всій ієрархії. Якщо ж використовується множинне спадкування, то можна зіткнутися зі специфічними проблемами для вирішення яких розробили алгоритм MRO C3. Суть

алгоритму – отримати лінійний список класів, які задають послідовність пошуку методів. Даний список зберігається у полі `__mro__` класу. Наприклад, для попередньої ієрархії класів `D.__mro__` містить список (`<class '__main__.D'>, <class '__main__.B'>, <class '__main__.C'>, <class '__main__.A'>, <class 'object'>`). Тобто, при виклику методу `foo()` його пошук спочатку відбуватиметься у класі D, потім у класі B і аж потім у класі C. Якщо він не буде знайдений, то згенерується помилка `Attribute Error`.

Бувають випадки коли MRO неможливо побудувати. Наприклад, при наступному оголошенні:

```
class C(A, B): pass
class D(B, A): pass
class E(C, D): pass
```

У даному випадку конфлікт залишається нерозв'язаним, оскільки в оголошені класу C клас A стоїть перед B, а у оголошенні класу D – навпаки. Для розв'язання цієї колізії є 2 способи:

1. Переглянути структуру класів, оскільки скоріш за все у ній є помилка. У даному прикладі достатньо буде поміняти послідовність класів A і B місцями у одному з класів C або D.
2. Створити метаклас перевизначивши метод `mro` класу `type`, який повертатиме потрібну послідовність класів і застосувати створений метаклас при оголошенні класу E.

```
class MetaMRO(type):
    def mro(cls):
        return (cls, A, B, C, D, object)

class E(C, D, metaclass = MetaMRO): pass
```

У Python до версії 2.2, класи неявно не спадкували клас `object` і MRO був простішим: пошук відбувався у всіх батьківських класах зліва напарво на максимальну глибину.

Класи-домішки

Домішки або Mixin – це шаблон проектування, в якому деякий метод базового класу використовує метод, який не визначається у цьому класі. Цей метод призначений для реалізації іншим базовим класом. Клас-домішка або mixin class – це клас, який використовується у цьому шаблоні, надаючи функціональні можливості (методи), але не призначений для самостійного використання у вигляді об'єктів класу. В ідеальному випадку класи-домішки не мають власної ієрархії спадкування і не мають полів, а мають лише методи.

У Python немає ніякого спеціального синтаксису для підтримки класів-домішок, тому їх легко сплутати зі звичайними класами, але при цьому між ними є велика різниця. Для того щоб позначити, що клас є класом-домішкою, використовують суфікс `Mixin` в кінці назви класу.

Класи-домішки використовуються при реалізації множинного спадкування і не лише у Python. Проте у Python реалізація множинного спадкування з використаням класів-домішок має свою специфіку пов'язану з MRO – класи-домішки мають бути вказані на початку списку базових класів, інакше методи класу-домішки можуть бути перекриті методами інших базових класів, що не є класами-домішками.

Розглянемо приклад множинного спадкування з використанням класу-домішки.

```
class Entity:
    def __init__(self, pos_x, pos_y):
        self.pos_x = pos_x
        self.pos_y = pos_y

class SquareMixin:
    def add_size(self, size_x):
        self.size_x = size_x
        self.size_y = size_x

    def perimeter(self):
        return self.size_x * 4

    def square(self):
        return self.size_x * self.size_x

class SquareEntity(SquareMixin, Entity):
    def print_square(self):
        print(f'Square size = {self.square()}')

--------------------------------------------------
square = SquareEntity(5, 4)
square.add_size(20)
square.print_square()
```

Тут похідний клас `SquareEntity()` отримує від класу-домішки `SquareMixin()` методи додавання розміру квадрата, а також обчислення його периметра та площі. Ця поведінка спрощує дерево спадкування `SquareEntity()`, що дозволяє використовувати клас `Entity()` як батьківський для інших фігур без необхідності успадковувати методи, які не потрібні (наприклад, для кола).

Якщо необхідно явно передати параметр size при ініціалізації класу `SquareEntity()` з прикладу вище, необхідно скористатися функцією `super()`.

```
class Entity:
    def __init__(self, pos_x, pos_y):
        self.pos_x = pos_x
        self.pos_y = pos_y

class SquareMixin:
    def __init__(self, size, **kwargs):
        super().__init__(**kwargs)
        self.size_x = size
        self.size_y = size

    def perimeter(self):
        return self.size_x * 4

    def square(self):
        return self.size_x * self.size_x


class SquareEntity(SquareMixin, Entity):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)

    def print_square(self):
        print(f'Square size = {self.square()}')


# Всі параметри передаються як пара ключ-значення
square = SquareEntity(pos_x=5, pos_y=4, size=20)
square.print_square()
```

КОНТРОЛЬНІ ПИТАННЯ

1. Що таке модулі?
2. Як імпортувати модуль?
3. Як оголосити клас?
4. Що може міститися у класі?
5. Як називається конструктор класу?
6. Як здійснити спадкування?
7. Які види спадкування існують?
8. Які небезпеки є при множинному спадкуванні, як їх уникнути?
9. Що таке класи-домішки?
10. Яка роль функції `super()` при спадкуванні?

ЛІТЕРАТУРА

1. Lutz M. Learning Python, 5th Edition / Mark Lutz. – O'Reilly, 2013. – 1643 p.
2. Васильєв О. Програмування мовою Python / Олексій Васильєв. – К: НК Богдан, 2019. – 504 с.
3. Python documentation [електронний ресурс]. – Режим доступу до документації: https://docs.python.org/3/
4. Python Enhancement Proposals [електронний ресурс]. – Режим доступу до документації: https://peps.python.org/

ЗАВДАННЯ

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.
4. Дати відповідь на контрольні запитання.

ВАРІАНТИ ЗАВДАННЬ

Базові класи

1. Людина
2. Космічний корабель
3. Пес
4. Кіт
5. Машина
6. Літак
7. Комп'ютер
8. Фотоапарат
9. Рослина
10. Будинок
11. Монітор
12. Водойма
13. Телефон
14. Телевізор
15. Корабель
16. Аудіоплеєр
17. Відеоплеєр
18. Сканер
19. Принтер
20. Взуття
21. Пістолет
22. Автомат
23. Плитка для приготування їжі
24. Спорядження альпініста
25. Кондиціонер
26. Шлюпка на веслах
27. Патрон
28. Лампочка
29. Протигаз
30. Локомотив

Похідні класи

1. Спортсмен
2. Багаторазовий космічний корабель
3. Піддослідний пес
4. Піддослідний кіт
5. Вантажна машина
6. Бомбардувальник
7. Ноутбук
8. Цифрова відеокамера

9. Дерево
10. Офісний центр
11. Сенсорний екран
12. Море
13. Мобільний телефон
14. Телевізор з тюнером
15. Фрегат
16. Диктофон
17. Відеомагнітофон
18. Копіювальний апарат
19. Багатофункціональний пристрій
20. Чоботи
21. Водяний Пістолет
22. Штурмова гвинтівка
23. Газова плитка
24. Спорядження військового альпініста
25. Пристрій кліматконтролю
26. Моторний човен
27. Інтелектуальний патрон
28. Енергозберігаюча лампочка
29. Протигаз командира
30. Електричка

ПОРЯДОК ВИКОНАННЯ

1. Переконайтеся, що у вас є встановлені інтерпретатор Python і середовище для розробки Microsoft Visual Studio Code (VS Code).
2. Створіть папку для проекту на диску.
3. Відкрийте середовище VS Code.
4. Відкрийте вкладку Extensions (Ctrl + Shift + X) і переконайтеся, що у VS Code встановлено модуль розширення під назвою “Python”.
5. Відкрийте створену на кроці 2 папку, вибравши її з меню “File” -> “Open Folder”.
6. Створіть нові Python модулі (файли з розширенням .py), вибравши в меню “File” -> “New File” -> “Python file” і вказавши почергово їх імена.
7. Скопіюйте код демонстраційної програми, що наведена в методичних вказівках, у відповідні новостворені модулі і збережіть їх.
8. Створіть віртуальне середовище Python, натиснувши Ctrl+Shift+P та вибравши зі списку команду “Python: Create Environment” -> “Venv” -> <шлях до встановленого інтерпретатора Python>. Після її виконання має появитися папка .venv у вашій робочій папці і вибраний інтерпретатор з віртуального середовища у нижньому правому куті рядка стану VS Code.
9. Дослідіть демонстраційну програму у відлагоджувачі.
10. Створіть власні модулі. Для цього повторіть крок 6 і збережіть новостворені модулі у робочій папці.
11. У створених файлах напишіть програму згідно завдання.
12. Запустіть програму на виконання. Для цього слід вибрати підпункт меню “Run” -> “Start Without Debugging”.
13. Встановіть точки переривання, запустіть налагоджувач (“Run” -> “Start Debugging”) та покроково дослідіть процес виконання програми.

ДЕМОНСТРАЦІЙНА ПРОГРАМА

Файл Scroller.py

```
################################################################################
# Copyright (c) 2023 Lviv Polytechnic National University. All Rights Reserved.
#
# This program and the accompanying materials are made available under the terms
# of the Academic Free License v. 3.0 which accompanies this distribution, and is
# available at https://opensource.org/license/afl-3-0-php/
#
# SPDX-License-Identifier: AFL-3.0
################################################################################
```

```
class Scroller:
    class Directions:
        NEUTRAL = u'NEUTRAL'
        UP = u'UP'
        DOWN = u'DOWN'

    def __init__(self):
        self.__direction = Scroller.Directions.NEUTRAL

    def setUpDirection(self):
        self.__direction = Scroller.Directions.UP

    def setNeutralDirection(self):
        self.__direction = Scroller.Directions.NEUTRAL

    def setDownDirection(self):
        self.__direction = Scroller.Directions.DOWN

    def resetScroller(self):
        self.setNeutralDirection()

    def getDirection(self):
        return self.__direction
```

Файл RelativePosition.py

```
#############################################################################
# Copyright (c) 2023 Lviv Polytechnic National University. All Rights Reserved.
#
# This program and the accompanying materials are made available under the terms
# of the Academic Free License v. 3.0 which accompanies this distribution, and is
# available at https://opensource.org/license/afl-3-0-php/
#
# SPDX-License-Identifier: AFL-3.0
#############################################################################

class RelativePosition:
    def __init__(self, xPos = 0, yPos = 0):
        self.__x = xPos
        self.__y = yPos

    def getXPosition(self):
        return self.__x

    def getYPosition(self):
        return self.__y

    def setXPosition(self, xPos):
        self.__x = xPos

    def setYPosition(self, yPos):
        self.__y = yPos
```

Файл Button.py

```
#############################################################################
# Copyright (c) 2023 Lviv Polytechnic National University. All Rights Reserved.
#
# This program and the accompanying materials are made available under the terms
# of the Academic Free License v. 3.0 which accompanies this distribution, and is
# available at https://opensource.org/license/afl-3-0-php/
#
# SPDX-License-Identifier: AFL-3.0
#############################################################################

class Button:

    def __init__(self, res=1000000):
        self.__btnResource = res
```

```
    def clickButton(self):
        self.__btnResource = self.__btnResource - 1

    def getButtonResource(self):
        return self.__btnResource
```

Файл ComputerMouse.py

```
##############################################################################
# Copyright (c) 2023 Lviv Polytechnic National University. All Rights Reserved.
#
# This program and the accompanying materials are made available under the terms
# of the Academic Free License v. 3.0 which accompanies this distribution, and is
# available at https://opensource.org/license/afl-3-0-php/
#
# SPDX-License-Identifier: AFL-3.0
##############################################################################

import Scroller
import RelativePosition
import Button

class ComputerMouse:
    def __init__(self, resource=1000000):
        self.__scrollerDevice = Scroller()
        self.__pos = RelativePosition()
        self.__rightButton = Button(resource)
        self.__leftButton = Button(resource)

    def moveMouseOnDistance(self, xPos, yPos):
        self.__pos.setXPosition(self.pos.getXPosition() + xPos)
        self.__pos.setYPosition(self.pos.getYPosition() + yPos)

    def setMousePosition(self, xPos, yPos):
        self.__pos.setXPosition(xPos)
        self.__pos.setYPosition(yPos)
        print(f"Mouse position: x={self.getMouseXPosition()}; y =
{self.getMouseYPosition()}.")

    def getMouseXPosition(self):
        return self.__pos.getXPosition()

    def getMouseYPosition(self):
        return self.__pos.getYPosition()

    def clickRightButton(self):
        self.__rightButton.clickButton()
        print("Right mouse button clicked")

    def clickLeftButton(self):
        self.__leftButton.clickButton()
        print("Left mouse button clicked")

    def scrollUp(self):
        self.__scrollerDevice.setUpDirection()
        print("Mouse scroll up")

    def scrollDown(self):
        self.__scrollerDevice.setDownDirection()
        print("Mouse scroll down")

    def resetScroller(self):
        self.__scrollerDevice.resetScroller()

    def getScrollingDirection(self):
        return self.__scrollerDevice.getDirection()
```

Файл GamingMouse.py

```
##############################################################################
# Copyright (c) 2023 Lviv Polytechnic National University. All Rights Reserved.
#
# This program and the accompanying materials are made available under the terms
# of the Academic Free License v. 3.0 which accompanies this distribution, and is
# available at https://opensource.org/license/afl-3-0-php/
#
# SPDX-License-Identifier: AFL-3.0
##############################################################################

import ComputerMouse

class GamingMouse(ComputerMouse):
    def __init__(self, weight, resource=1000000):
        super().__init__(resource)
        self.__fun1Button = Button(resource)
        self.__fun2Button = Button(resource)
        self.__weight = weight

    def getWeight(self):
        return self.__weight

    def clickFunction1Button(self):
        self.__fun1Button.clickButton()
        print("Function 1 mouse button clicked")

    def clickFunction2Button(self):
        self.__fun2Button.clickButton()
        print("Function 2 mouse button clicked")
```

Файл Main.py

```
##############################################################################
# Copyright (c) 2023 Lviv Polytechnic National University. All Rights Reserved.
#
# This program and the accompanying materials are made available under the terms
# of the Academic Free License v. 3.0 which accompanies this distribution, and is
# available at https://opensource.org/license/afl-3-0-php/
#
# SPDX-License-Identifier: AFL-3.0
##############################################################################

import GamingMouse

if __name__ == "__main__":
    mouse = GamingMouse(150)
    mouse.clickLeftButton()
    mouse.clickRightButton()
    mouse.setMousePosition(5, -3)
    mouse.scrollUp()
    dir = mouse.getScrollingDirection()
    if dir == Scroller.Directions.DOWN:
        print("Scrolling direction: Down")
    elif dir == Scroller.Directions.UP:
        print("Scrolling direction: Up")
    else:
        print("Scrolling direction: Neutral")
    mouse.clickFunction1Button()
    mouse.clickFunction2Button()
```

НАВЧАЛЬНЕ ВИДАННЯ

ОСНОВИ ОБ’ЄКТНО-ОРІЄНТОВАНОГО ПРОГРАМУВАННЯ У PYTHON

МЕТОДИЧНІ ВКАЗІВКИ

до лабораторної роботи № 9 з дисципліни “ Кросплатформні засоби програмування ”
для студентів базового напряму 6.123 - “Комп’ютерна інженерія”

Укладач Олексів М. В., к.т.н.

Редактор

Комп’ютерне верстання

Здано у видавництво . Підписано до друку
Формат 70х100/16. Папір офсетний. Друк на різографі
Умовн. друк. арк. Обл..-вид. арк..
Тираж прим. Зам..

Видавництво Національного університету “Львівська політехніка”
Реєстраційне свідоцтво ДК №751 від 27.12.2001 р.

Поліграфічний центр Видавництва
Національного університету “Львівська політехніка”

Вул. Ф. Колесси, 2. Львів, 79000