Лабораторна робота № 14

Тема: Python ООП. **Класи.**

Мета: Отримати практичні навички використання класів Python.

Література: Васильєв О.М. Програмування мовою Python – Тернопіль: Навчальна книга – Богдан, 2019. – 504с. стор. 301-359

https://proglib.io/p/python-oop

https://pythonru.com/primery/primery-raboty-s-klassami-v-python

Зміст роботи:

Написати програми для рішення задач:

Завдання 1.

Оголосіть клас Point3D для точок із трьома координатами x, y, z. Створіть кілька екземплярів цього класу і через них виведіть у консоль значення x, y, z. Далі, зробіть такі маніпуляції:

- поміняйте будь-яке значення координати в класі Point3D і подивіться як це вплине на відображені величини екземплярів класу;
- видаліть координату z у класі Point3D і переконайтеся, що вона буде відсутня у всіх примірниках;
- Поміняйте координату в будь-якому екземплярі класу і подивіться на результат.

Завдання 2.

Створіть клас Student. Опишіть метод name що визначає ПІБ студента, метод marks який визначає оцінки з дисциплін і метод average_marks що обчислює середнє значення.

Завдання 3.

Створіть клас **Triangle**, що буде містити дані про трикутник. Опишіть методи обчислення периметра та площі цього трикутника. Клас **Triangle**, буде містити три поля – довжини сторін трикутника – a, b, c. Опишіть метод **perimeter**, що визначає периметр трикутника та метод **square**, що обчислює площу зазначеного трикутника використовуючи формулу Герона. (використати правило перевірки існування трикутника)

Методичні рекомендації

Визначення класу:

```
class ім'я_класу:
        інструкція 1
        ....
        інструкція N
```

Створення об'єкта класу:

```
 об'єкт_класу = ім'я_класу()
```

Класи збирають в собі набори даних (змінних) разом з наборами функцій, що на них діють. Мета полягає в тому, щоб досягти більш модульного коду за допомогою групування змінних і функцій, в невеликі вузли, що легко модифікувати.

За згодою у Python для посилання на об'єкт використовується ім'я self. Змінна self зв'язується з об’єктом, до якого було застосовано даний метод, і через цю змінну ми отримуємо доступ до атрибутів об'єкта. Коли цей же метод застосовується до іншого об'єкта, то self зв'яжеться вже з саме цим іншим об'єктом, і через цю змінну будуть викликатись тільки його поля.

```
class Adder:
        n = 1
        def add(self, v):
                return v + self.n
a = Adder()
b = Adder()
a.n = 10
print(a.add(3))
print(b.add(4))
```

Тут від класу Adder створюється два об'єкта – a та b. Для об'єкта a заводиться власне поле n. Об'єкт b, не має такого поля, отже успадковує n від клас Adder.

У методі add() вираз self.n – це звернення до поля n, переданого об'єкта, і не важливо, на якому рівні наслідування його буде знайдено.

Необхідність конструкторів пов'язана з тим, що часто об'єкти повинні мати власні властивості одразу. Припустимо маємо клас Person, об'єкти котрого обов'язково повинні мати ім'я. Якщо клас буде описано наступним способом:

```
        class Person():
                def set_name(self, name):
                        self.name = name
```

то створення об'єкта можливе без полів. Для встановлення імені метод

set_name() необхідно викликати окремо:

```
p1 = Person()
p1.name = 'Jane Doe'
print(p1.name)
'Jane Doe'
```

Наявність конструктора не дозволить створити об'єкт без полів:

```
class Person():
    def __init__(self, name):
        self.name = name
p1 = Person('Jane Doe')
print(p1.name)
```

Тут при виклику класа в круглих дужках передаються значення, котрі будуть присвоєні параметрам метода init(). Перший параметр – self – посилання на сам щойно створений об'єкт.

Контрольні запитання.

1. Що таке клас?
2. Які існують атрибути класу?
3. Як створити клас?
4. В чому полягає призначення методів класу?
5. Як ініціалізувати поля класу?
6. Як працює конструктор класу?
7. Як працює деструктор класу?
8. Що таке об’єкт класу?
9. Як створити об’єкт (екземпляр) класу?
10. В чому полягає призначення змінної self?