Python

ООП

План

1. Поняття ООП
2. Створення класів
3. Створення екземплярів класів
4. Інкапсуляція
5. Конструктори і «магічні» методи в Python

Парадигми програмування

Парадигма програмування - це сукупність ідей і понять, що визначають стиль написання комп'ютерних програм, підхід до програмування.

Python підтримує різні парадигми програмування

- імперативне програмування
  - процедурне програмування
  - структурне програмування
- об'єктно - орієнтоване програмування
- функціональне програмування

Об'єктно-орієнтоване програмування

Об'єктно-орієнтоване програмування (ООП) - парадигма програмування, в якій основними концепціями є поняття об'єктів і класів.

Клас є моделлю ще не існуючої сутності (об'єкта). Він є складовим типом даними, що включає в себе поля і методи.

Об'єкт - це екземпляр класу.

Основні принципи ООП:

- Абстракція
- Інкапсуляція
- Поліморфізм
- Спадкування

Інкапсуляція

Інкапсуляція - це властивість системи, що дозволяє об'єднати дані і методи, що працюють з ними, в класі, і <u>приховати</u> деталі реалізації.

Інкапсуляція забезпечується наступними засобами:

- контроль доступу
- методи доступу
- властивості об'єкта

ООП в Python

В Python все є об'єктами - екземплярами яких-небудь класів, навіть самі класи, які є об'єктами - екземплярами метакласів. Головним метакласом є клас type, який є абстракцією поняття типу даних.

Класи в Python

- У термінології Python члени класу називаються атрибутами. Ці атрибути можуть бути як змінними, так і функціями.
- Класи створюються за допомогою ключового слова class.
- Класи як об'єкти підтримують два види операцій: звернення до атрибутів класів і створення (інстанціонування) об'єктів - екземплярів класу (instance objects).
- Звернення до атрибутів якого -або класу або об'єкта проводиться шляхом вказівки імені об'єкта і назви ознаки через крапку.
- Для створення екземплярів класу використовується синтаксис виклику функції.

Екземпляри класів в Python

- Єдина доступна операція для об'єктів-екземплярів - це доступ до їх атрибутів.
- Атрибути *об'єктів-екземплярів* діляться на два типи: атрибути-дані і методи.
- Атрибути-дані аналогічні полям у термінології більшості широко поширених мов програмування.
- Атрибути-дані не потрібно описувати: як і змінні, вони створюються в момент першого присвоювання. Як правило, їх створюють в методі - конструкторі __init__.
- Метод - це функція, що належить об'єкту. Всі атрибути класу, які є функціями, описують відповідні методи його екземплярів, однак вони не є одним і тим же.
- Особливістю методів є те, що в якості першого аргументу вони обидві приймають даний екземпляр класу. Таким чином, якщо obj - екземпляр класу MyClass, виклик методу obj.method () відповідає виклику функції MyClass.method (obj).

Приклад 1

```
class MyClass:
    int_field = 8
    str_field = 'a string'

# Звертання до атрибутів класу
print(MyClass.int_field)
print(MyClass.str_field)

# Створення екземплярів класу
object1 = MyClass()
object2 = MyClass()

# Звертання до атрибутів класу
    через його екземпляри
print(object1.int_field)
print(object2.str_field)
```

```
# Зміна значення атрибуту класу
MyClass.int_field = 10
print(MyClass.int_field)
print(object1.int_field)
print(object2.int_field)

object1.str_field = 'another string'
print(MyClass.str_field)
print(object1.str_field)
print(object2.str_field)
```

```
8
a string
8
a string
10
10
10
a string
another string
a string
```

Різниця між атрибутами класу і атрибутами - даними

- Атрибути класу є загальними для самого класу і всіх його екземплярів. Їх зміна відображається на всі відповідні об'єкти. Атрибути-дані належать конкретному екземпляру і їх зміна ніяк не впливає на відповідні атрибути інших екземплярів даного класу. Таким чином, атрибути класу, які не є функціями, приблизно відповідають статичним полям в інших мовах програмування, а атрибути-дані - звичайним полям.

Приклад 2

```
class Person:
    pass

alex = Person()
alex.name = 'Alex'
alex.age = 18

john = Person()
john.name = 'John'
john.age = 20

print(alex.name, 'is', alex.age)
print(john.name, 'is', john.age)
```

```
Alex is 18
John is 20
```

Приклад 3

```
class Person:

    def print_info(self):
        print(self.name, 'is', self.age)

alex = Person()
alex.name = 'Alex'
alex.age = 18

jon = Person()
jon.name = 'John'
jon.age = 20

Person.print_info(alex)
jon.print_info()

print(type(Person.print_info))
print(type(jon.print_info))
```

```
Alex is 18
Jon is 20
<class 'function'>
<class 'method'>
```

```
>>> Person.print_info
<function Person.print_info at 0x03254C90>
>>> jon.print_info
<bound method Person.print_info of
<__main__.Person object at 0x034CA930>>
```

Приклад 4

```
class Person:

# Конструктор
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def print_info(self):
        print(self.name, 'is', self.age)

alex = Person('Alex', 18)
john = Person('John', 20)

alex.print_info()
john.print_info()
```

Alex is 18
Jon is 20

Статичні методи і методи класу

- **Декоратор - це спеціальна функція, яка змінює поведінку функції або класу. Для застосування декоратора слід перед відповідним оголошенням вказати символ @, ім'я необхідного декоратора і список його аргументів в круглих дужках. Якщо передача параметрів декораторові не потрібна, дужки не вказуються.**
- **Для створення статичних методів використовується декоратор staticmethod.**
- **Для створення методів класу використовується декоратор classmethod.**

Методи класу схожі на звичайні методи, які відносяться до самого класу як до об'єкту - екземпляру метакласу. Їх перший аргумент прийнято називати cls.

на відміну від :

- **Звичайні методи належать об'єктам - екземплярів класів**
- **Статичні методи, які відносяться до самого класу і до всіх його екземплярів але не належать жодному об'єкту - екземпляру).**

Приклад 5

8
42
8

```
class MyClass:
    # Оголошення атрибуту класа
    class_attribute = 8

    # Конструктор
    def __init__(self):
        self.data_attribute = 42

    # Статичний метод
    @staticmethod
    def static_method():

     print(MyClass.class_attribute)

    # Звичайний метод
    def instance_method(self):
        print(self.data_attribute)

if __name__ == '__main__':
    # Виклик статичного методу
    MyClass.static_method()
    # Інстанціонування об'єкта
    obj = MyClass()
    # Виклик методу
    obj.instance_method()
```

Приклад 6

```
class Rectangle:

    def __init__(self, side_a, side_b):
        self.side_a = side_a
        self.side_b = side_b

    def __repr__(self):
        """
        Метод, що повертає рядкове представлення об'єкту
        """
        return "Rectangle(%.1f, %.1f)" % (self.side_a, self.side_b)
```

Приклад 6

```
class Circle:

    def __init__(self, radius):
        self.radius = radius

    def __repr__(self):
        return 'Circle(%.1f)' % self.radius

    @classmethod
    def from_rectangle(cls, rectangle):
        radius = (rectangle.side_a ** 2 + rectangle.side_b ** 2) ** 0.5 / 2
        return cls(radius)
```

```
@staticmethod
def from_rectangle(rectangle):
    radius = (rectangle.side_a ** 2 + rectangle.side_b ** 2) ** 0.5 /
2
    return Circle(radius)
```

Приклад 6

```
def main():
    rectangle = Rectangle(3, 4)
    print(rectangle)
    circle1 = Circle(1)
    print(circle1)
    circle2 = Circle.from_rectangle(rectangle)
    print(circle2)


if __name__ == '__main__':
    main()
```

```
Rectangle(3.0, 4.0)
Circle(1.0)
Circle(2.5)
```

Інкапсуляція в Python

- Усі атрибути за замовчуванням є **публічними**.
- Атрибути, імена яких починаються з *одного знака підкреслення* (_) говорять програмісту про те, що вони належать до внутрішньої реалізації класу і не повинні використовуватися ззовні, проте ніяк **не захищені**.
- Атрибути, імена яких починаються, але не закінчуються, *двома символами підкреслення*, вважаються **приватними**. До них застосовується механізм «**name mangling**». Він не передбачає захисту даних від зміни ззовні, так як до них все одно можна звернутися, знаючи ім'я класу, проте дозволяє захистити їх від **випадкового перевизначення в классах- нащадках**.

Спеціальні атрибути і методи

- Атрибути, імена яких починаються і закінчуються *двома знаками підкреслення*, є внутрішніми для Python і задають особливі властивості об'єктів (приклади: **__doc__, __class__**).
- Серед таких атрибутів є методи. У документації Python подібні методи називаються **методами зі спеціальними іменами**, проте в співтоваристві Python - розробників дуже поширена назва «**магічні методи**». Також, зустрічається і назва «**спеціальні методи**». Вони задають особливу поведінку об'єктів і дозволяють перевизначити поведінку вбудованих функцій і операторів для примірників даного класу.
- Найчастіше використовується метод **__init__,** який автоматично викликається після створення екземпляра класу.

Не слід оголошувати свої власні (нестандартні) атрибути з іменами, які починаються і закінчуються двома знаками підкреслення