Microsoft Partner
Silver Learning

Python Essential

ООП - Класи, атрибути, методи, конструктор

ITVDN
IT VIDEO DEVELOPERS NETWORK

План заняття

1. Що таке парадигма програмування?
2. Що таке ООП?
3. Основні принципи ООП.
4. Що таке клас?
5. Що таке метод?
6. Що таке об’єкт?
7. Реалізація в Python

Після уроку обов'язково

Повторіть цей урок у відео форматі на ITVDN.com

Перевірте, як Ви засвоїли даний матеріал на TestProvider.com

Тема

Введення в об’єктно-орієнтоване програмування

Парадигми програмування

Парадигма програмування – це сукупність ідей та 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

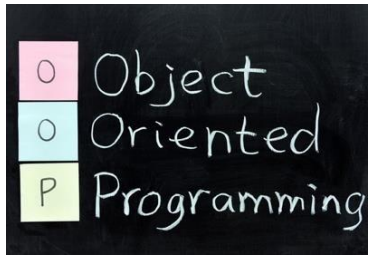
Об'єктно-орієнтоване програмування

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

Клас є моделлю ще не існуючої сутності (об’єкту). Він є складовим типом даних, що включає в себе поля та методи.
Об'єкт – це екземпляр класу.

Основні принципи ООП:

- Абстракція
- Інкапсуляція
- Поліморфізм
- Наслідування



ООП в Python

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

Усе є об‘єкт

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 та створення (*інстанціювання*) об'єктів – **екземплярів класу** (instance objects).
- Звернення до атрибутів якого-небудь класу чи об'єкту здійснюється шляхом вказування імені об'єкта та імені атрибуту через крапку.
- Для створення екземплярів класу використовується синтаксис виклику функції.

Поля	Методи
собака.вік	собака.бігти()
собака.колір_шерсті	собака.плисти()
кіт.колір_шерсті	кіт.лазити_по_деревам()
риба.вид	риба.плисти()
тамагочі.ім’я	тамагочі.спати()
тамагочі.вік	тамагочі.їсти()

Клас

Клас – це конструкція мови, котра складається з ключового слова class, ідентифікатора (імені) та тіла.
Клас може містити в своєму тілі: *поля*, *методи*.
Поля визначають *стан*, а ***методи*** — *поведінку* майбутнього об'єкту.

```
class Car:

    my_var = 5
4 пробіли або 1 Tab
    def my_method(self):
        print("Hello! I'm a new method this class!")

4 пробіли або 1 Tab
```

Екземпляри класів у Python

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

Перший аргумент методу, котрий відповідає поточному екземпляру, прийнято називати *self*.

Різниця між атрибутами класу та атрибутами-даними

Атрибути класу є спільними для самого класу та всіх його екземплярів. Їхня зміна відображається на всі відповідні об'єкти. **Атрибути-дані належать конкретному екземпляру та 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.

Слід розуміти різницю між атрибутами класу та атрибутами-даними

Object and instances

Об’єкти містять в собі статичні поля та всі методи. **Екземпляри містять** динамічні поля.

```
class Airplane:
    # Атрибут класу Airplane
    my_count = 0

    # Метод класу Airplane
    def start(self, model, color, number, price):
        self.model = model
        self.color = color
        self.number = number
        self.price = price
        print(f"Hello! I'm a airplane series {self.model}, {self.color}, my tail number is {self.number}, my cost is {self.price}$")
        Airplane.my_count += 1
```

Об’єкт

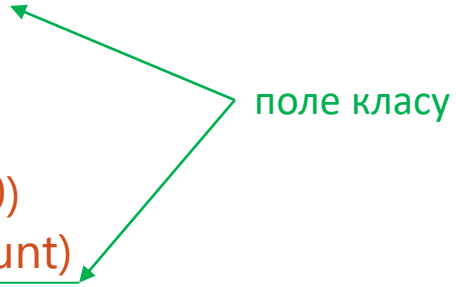
Object and instances

Екземпляр 1 = Клас()

```
ap1 = Airplane()
ap1.start("Airbus S.A.S", 426, "white", 10000000)
print("Всього в ангарі літаків:", ap1.my_count)
```

Екземпляр 2 = Клас()

```
ap2 = Airplane()
ap2.start("Boeing", 236, "metallic", 17000000)
print(" Всього в ангарі літаків :", ap2.my_count)
```



Конструктор класу

Конструктор класу – спеціальний метод, який викликається під час побудови класу.

Конструктори бувають двох видів:

Конструктори за замовчуванням

Користувацькі конструктори

На відміну від інших популярних об'єктно-орієнтованих мов програмування, таких як Java, C#, мова Python не підтримує кілька конструкторів, тобто у разі визначення кількох методів __init __(), останній перевизначить більш ранні визначення.

Якщо в тілі класу не визначено явно жодного конструктора, завжди використовується «невидимий» конструктор за замовчуванням. Для визначення конструктора використовується магічний метод __init__(). Конструктори не мають значень, що повертаються.

Конструктор класу

Завдання конструктора за замовчуванням – ініціалізація полів за замовчуванням.

Завдання користувацького конструктора — ініціалізація полів обумовленими користувачем значеннями.

Якщо в класі визначено користувацький конструктор, і при цьому потрібно створювати екземпляри класу з використанням конструктора за замовчуванням, то конструктор за замовчуванням повинен бути визначений в тілі класу явно, інакше виникне помилка під час виконання програми.

Магічний метод __new__

- Для створення об'єкту при створенні екземпляра класу, в Python початково викликається метод __new__(), після якого викликається метод __init__() для ініціалізації атрибутів об'єкту.
- __new__() — це статичний метод класу об’єкту. Він виглядає наступним чином:

object.__new__(class, *args, **kwargs), де:

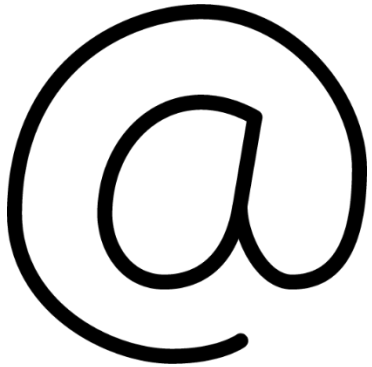
– class — це клас нового об’єкту, котрий необхідно створити.

– параметри *args и **kwargs мають співпадати з параметрами конструктора класу, однак метод __new__() їх використовує.

Метод __new__() повинен повертати новий об'єкт класу, але це не обов'язково. При визначенні нового класу цей клас явно успадковується від класу об'єкта. Це означає, що ви можете перевизначити статичний метод __new__() і зробити щось до та після створення нового екземпляра класу. Щоб створити об'єкт класу, ви викликаєте метод super().__new__().

Статичні методи та 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 та список його аргументів у круглих дужках. Якщо передача параметрів декоратору 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, але відносяться до самого класу як до об'єкта-екземпляра метакласу (на відміну від звичайних методів, котрі належать об'єктам-екземплярам класів, та статичних методів, які відносяться до самого класу та всіх його екземплярів і не належать жодному об'єкту-екземпляру). Їх перший аргумент прийнято називати *cls*.

Спеціальні атрибути та методи

- Атрибути, імена яких починаються та 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**, однак у спільноті Python-розробників дуже поширена назва "**магічні методи**". Також зустрічається і назва «**спеціальні методи**». Вони задають особливу поведінку об'єктів і дозволяють перевизначати поведінку вбудованих функцій та операторів для екземплярів даного класу.
- Початково буде викликатися метод __new__, який викличеться в момент ініціалізації об’єкту.
- Найчастіше використовуваним зі спеціальних методів є метод __init__, який автоматично викликається після створення екземпляра класу.
- Метод __del__, за допомогою якого можна визначати поведінку об'єктів, коли вони знаходяться у збирачі сміття (сокети, файлові об'єкти).

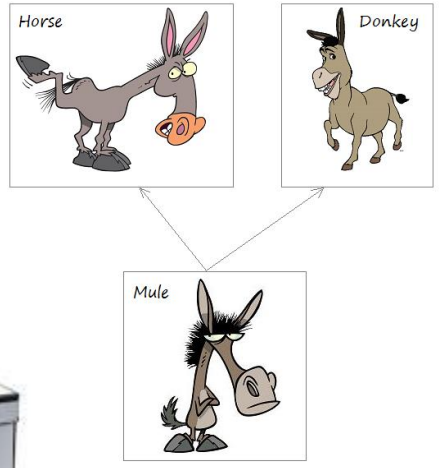
Не слід оголошувати власні (нестандартні) атрибути з іменами, які починаються і закінчуються двома знаками підкреслення

Основні принципи ООП

Абстракція – це моделювання необхідних атрибутів і взаємодій сутностей у вигляді класів для визначення абстрактного представлення системи.

Інкапсуляція – це приховування внутрішнього стану і функцій об'єкта та надання доступу лише через відкритий набір функцій.

Наслідування – це можливість створення нових абстракцій на основі існуючих.



Поліморфізм – можливість реалізації успадкованих властивостей або методів різними способами в рамках безлічі абстракцій.

Інкапсуляція в Python

- Усі атрибути за замовчуванням є **публічними**.
- Атрибути, імена яких починаються з *одного символу підкреслення* (_) говорять програмісту про те, що вони відносяться до внутрішньої реалізації класу і не повинні використовуватися ззовні, але 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, тому що до них все одно можна звернутися, знаючи ім'я класу і те, як Python змінює імена приватних атрибутів, проте *дозволяє захистити їх від випадкового перевизначення у класах-нащадках.*

Інкапсуляція

Інкапсуляція – це властивість системи, що дозволяє *об'єднати* дані та методи, що працюють з ними, у класі, і *приховати* деталі реалізації.

Інкапсуляція забезпечується наступними засобами:

- контроль доступу;
- методи доступу;
- властивості об'єкту.

Використання модифікаторів доступу до Python

Існують 3 типи модифікаторів доступів в ООП Python, які визначають видимість членів класу:

публічний — public (за замовчуванням);

приватний — private з використанням __name_attr;

захищений — protected з використанням _name_attr.

Ніколи не слід робити поля відкритими — це поганий стиль. Рекомендується використовувати методи доступу для звернення до поля.

Дивіться наші уроки у відео форматі

ITVDN.com

Перегляньте цей урок у відео форматі на освітньому порталі ITVDN.com
для закріплення пройденого матеріалу.

Усі курси записані сертифікованими тренерами, які працюють у навчальному центрі CyberBionic Systematics

Перевірка знань

TestProvider.com

TestProvider – це online сервіс перевірки знань з інформаційних технологій. З його допомогою Ви можете оцінити Ваш рівень та виявити слабкі місця. Він буде корисним як у процесі вивчення технології, так і для загальної оцінки знань IT-спеціаліста.

Після кожного уроку проходьте тестування для перевірки знань на TestProvider.com

Успішне проходження фінального тестування дозволить Вам отримати відповідний Сертифікат.

Q&A

Дякую за увагу!

Інформаційний відеосервіс для розробників програмного забезпечення