Advanced classes

Композиція та агрегація

- Композиція і агрегація є спеціалізованою формою асоціації.
- Тоді як асоціація - це відносини між двома класами без будь-яких правил.
- В даному відношенні повинен існувати клас контейнер який містить в собі об'єкт іншого класу

Приклад композиції

```
class Page:
    pass


class Album:
    def __init__(self, pages_num):
        self.pages = []
        for _ in pages_num:
            self.pages.append(Page())


album = Album()
```

Приклад агрегації

```
class GraphicsCard:
    pass


class Computer:
    def __init__(self, graphics_card):
        self.graphics_card = graphics_card


gc = GraphicsCard()
comp = Computer(gc)
```

Відмінність

- Отже відмінність в тому, де створений об'єкт що зберігається в класі контейнері.
- В першому випадку об'єкт створюється в класі, в другому випадку поза класом. Отже для першого випадку, видалення контейнера означає видалення і всіх компонентів цього контейнера. Коли в другому випадку ні.

build-in decorators

- classmethod
- staticmethod
- property

classmethod

- Метод класу — це метод, який прив’язаний до класу, а не до його об’єкта. Тобто він не вимагає створення самого об'єкта.

Приклад

```
class MyClass:
    @classmethod
    def method(cls):
        pass

obj = MyClass()

obj.method()
MyClass.method()
```

Де використовувати

- Використовувати можна для виклику інших методів класу, або статичних методів.
- Часто використовується як фабричний метод

Приклад фабричного методу

```
class User:
    def __init__(self, name, age, city):
        self.name = name
        self.age = age
        self.city = city

    @classmethod
    def from_config(cls, name, config):
        name, age, city = parse_config(config)
        return cls(name, age, city)
```

staticmethod

- Статичні методи, подібно до classmethod, є методами, які прив’язані до класу, а не до його об’єкта.
- Вони не вимагають створення екземпляра класу. Отже, вони не залежать від стану об'єкта.
- Різниця між статичним методом і методом класу така:
  - Статичний метод нічого не знає про клас і має справу лише з параметрами.
  - Метод класу працює з класом, оскільки його параметром завжди є сам клас.

Приклад

- ```
  class Calc:
      @staticmethod
      def sum(a, b):
          return a + b
  ```
  ```

- По суті статичні методи, це звичайні функції які інкапсульовані в клас.
- Зазвичай використовується для об'єднання декількох функцій за логічним змістом.

property

- Даний декоратор дає можливість створити property метод.
- Це метод який веде себе як властивість.

Приклад

```
class User:
    def __init__(self, name, surname):
        self.name = name
        self.surname = surname

    @property
    def full_name(self):
        return f"{self.name} {self.surname}"


user = User("Misha", "Klimchuk")
print(user.full_name)
```

- В даному прикладі `full_name` веде себе як властивість об'єкта. Тобто нам не потрібно викликати його як звичайний метод

Приклад сеттера

```
class User:
    def __init__(self, name, surname, card_number):
        self.name = name
        self.surname = surname
        self.__card_number = card_number

    @property
    def card_number(self):
        secret = "*" * (len(self.__card_number) - 4)
        return f"{secret}{self.__card_number[-4:]}"

    @card_number.setter
    def card_number(self, value):
        self.__card_number = value


user = User("Misha", "Klimchuk", "1234123412344321")
print(user.card_number)
user.card_number = "4321432143214444"
print(user.card_number)
```

- В даному прикладі, для виведення на екран номера карти, ми дещо змінили номер карти, показавши тільки останні 4 цифри.
- Декоратор `@card_number.setter` дав можливість зробити метод який запише значення номера карти в private атрибут

Slots

- __slots__ Дозволяє знизити обсяг пам'яті, що споживається екземплярами класу, обмежуючи кількість атрибутів, які вони підтримують. За замовчуванням класи використовують словник для зберігання атрибутів - це дозволяє модифікувати набір атрибутів об'єкта прямо під час виконання програми.

Динамічне додавання атрибута

```
class MyClass:
    pass
```

```
>>> obj = MyClass()
>>> obj.value = 10
>>> obj.value
10
```

Додавання атрибуту якого немає в слотах

```
class MyClass:
    __slots__ = ("name", )
>>> obj = MyClass()
>>> obj.value = 10
>>> obj.value
AttributeError: 'MyClass' object has no attribute 'value'
>>> obj.name  = "Stepan"
>>> obj.name
"Stepan"
```

Dataclass

- Датаклас, це клас який містить в собі тільки дані і не має додаткової функціональності

Приклад

```
import dataclasses


@dataclasses.dataclass
class Article:
    topic: str
    contributor: str
    language: str


article = Article("DataClasses", "misha", "EN")
article2 = Article("DataClasses_new", "mykola", "UA")
```

FrozenDataclass

- Датаклас в який не можна добавити нового поля
- Для оголошення такого датакласу потрібно передати аргумент frozen=True в декоратор

```
@dataclasses.dataclass(frozen=True)
```

NamedTuple

- NamedTuple — це клас, який містить дані у форматі словника, що зберігається в модулі collections.
- Доступ до даних здійснюється за допомогою певного ключа або індексе.

Приклад

- ```
  import collections

  User = collections.namedtuple('User', ['name',
  'age', 'city'])

  user_1 = Contributor("Misha", 25, "Kyiv")
  ```
  ```

Абстрактні класи

- Абстрактний клас можна розглядати як план (шаблон) для інших класів.
- Абстрактний клас дозволяє нам створити набір методів, які повинні бути обов'язково реалізовані в будь-яких дочірніх класах, побудованих з абстрактного класу.
- Використовуємо коли ми хочемо забезпечити загальний інтерфейс для різних реалізацій компонента.
- Клас, який містить один або кілька абстрактних методів, називається абстрактним класом.

Абстрактний метод

- Абстрактний метод — це метод, який має оголошення, але не має реалізації.

Приклад

```
from abc import ABC, abstractmethod

class MyABC(ABC):
    @abstractmethod
    def method(self, coordinate):
        pass
```

- Для створення абстрактного класу, наслідуємося від класу ABC, модуля abc.
- Також щоб клас став по справжньому абстрактним, потрібно добавити в нього абстрактний метод.
- Абстрактний метод робиться за допомогою додавання декоратора, `abstractmethod`.

abstractstaticmethod, abstractclassmethod, abstractproperty

- Дані декоратори можна використовувати для створення абстрактних - методів класу, статичних методів чи проперті.
- Проте в останніх версіях abc це вважається застарілим методом. Замість цього пропонується використання тільки декоратора `abstractmethod`

Приклад

```
class MyABC(ABC):
    @staticmethod
    @abstractmethod
    def method_2(coordinate):
        pass

    @classmethod
    @abstractmethod
    def method_3(cls):
        pass

    @property
    @abstractmethod
    def property_1(self):
        pass
```

Створення класу на основі абстрактного

- Для того щоб створити клас на основі абстрактного. Потрібно наслідуватись від абстрактного класу + перевизначити всі абстрактні методи

Приклад

```
class MyABC(ABC):
    @abstractmethod
    def method(self):
        pass


class MyClass(MyABC):
    def method(self):
        print("New method")
```