

Лабораторна робота 1

Знайомство з класами та об'єктами у python

Виконала:
студентка групи МІТ-31
Півторак Каріна
Варіант - 7

Мета: ознайомитися з парадигмою об'єктно - орієнтованого програмування, розглянути поняття класу та об'єкта, навчитися створювати класи та об'єкти

Завдання: самостійно опишіть клас згідно свого варіанту завдання. Створіть декілька екземплярів класу, продемонструйте функціонування усіх описаних у ньому методів

Хід роботи

Репозиторій: <https://github.com/KariSpace/python-oop-labs>

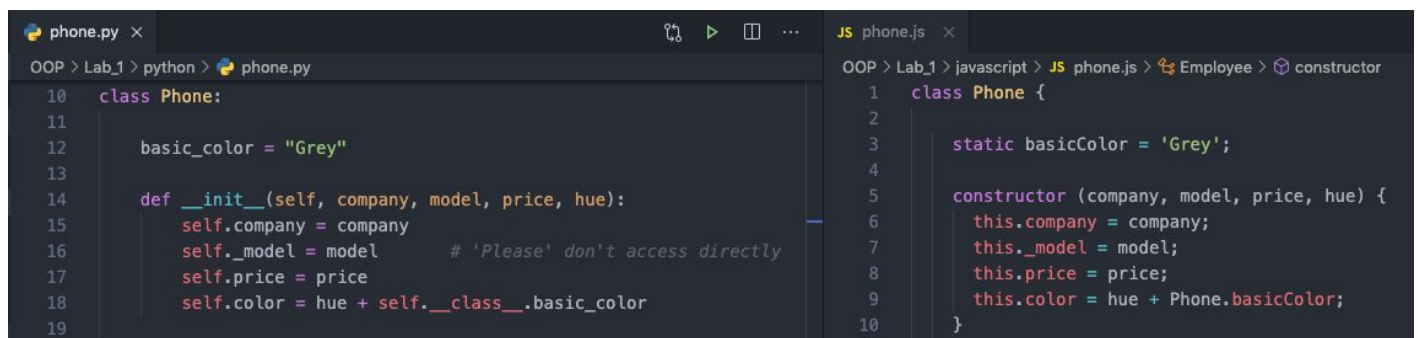
Для виконання цієї роботи було обрано мови Python та JavaScript. Це дві мови із динамічною типізацією, але абсолютно різним підходом до створення програм.

Python - мультипарадигменна мова з підтримкою об'єктно-орієнтованого програмування.

JavaScript використовує об'єктно-орієнтовану модель, яка не використовує класи. Ця об'єктно-орієнтована модель відома як програмування на основі прототипів

Створюємо клас Phone. У ньому є два типи змінних.

- **basic_color** (**basicColor** для JS) - статична (тобто, однакова для всіх екземплярів класу)
- **company, model, price** - динамічна
- **color** - динамічна, але формується із статичної змінної **basic_color** та динамічної **hue**



The screenshot shows a code editor with two tabs: 'phone.py' and 'phone.js'. The left pane displays the Python implementation of the 'Phone' class, and the right pane displays the JavaScript implementation.

```
phone.py
class Phone:
    basic_color = "Grey"

    def __init__(self, company, model, price, hue):
        self.company = company
        self._model = model # 'Please' don't access directly
        self.price = price
        self.color = hue + self.__class__.basic_color

phone.js
class Phone {
    static basicColor = 'Grey';

    constructor (company, model, price, hue) {
        this.company = company;
        this._model = model;
        this.price = price;
        this.color = hue + Phone.basicColor;
    }
}
```

Python:

```
class Phone:

    basic_color = "Grey"

    def __init__(self, company, model, price, hue):
        self.company = company
        self._model = model          # 'Please' don't access directly
        self.price = price
        self.color = hue + self.__class__.basic_color
```

JS

```
class Phone {

    static basicColor = 'Grey';

    constructor (company, model, price, hue) {
        this.company = company;
        this._model = model;
        this.price = price;
        this.color = hue + Phone.basicColor;
    }
}
```

Js і Python 3 надають 3 рівня доступу до даних:

- публічний (public, немає особливого синтаксису);
- захищений (protected)
 - Для пайтон- одне нижнє підкреслення на початку назви, `__protectedBanana`). Може змінюватись. можна описати як “будь-ласка, не чіпайте мене безпосередньо”
 - Для JS єдиний спосіб це - використання класу, який має геттер для властивості без сеттера
- приватний (private) , два нижніх підкреслення в початку назви, `__privateBanana`) - для пайтон. `accelerate`: - для Джаваскрипт

```

class Phone:

    color = "Grey"

    def __init__(self, company_name, model_name, price, color_name):
        self.company_name = company_name
        self._model_name = model_name          # 'Please' don't access
directly
        self.price = price
        self.color_name = color_name + self.__class__.color

    def __str__(self): # magic method / initializer
        """Return a descriptive string for this instance, invoked by
print() and str()"""
        return f'\nThis is a {self.color_name} {self.company_name}
{self._model_name} phone'

```

Перевантаження операторів в Python - це можливість за допомогою спеціальних методів в класах перевизначати різні оператори мови.

Для пайтон:

`__str__` - магічний метод. Він перевантажений для цього класу.

Для джава скрипт зробити це складніше:

```

Phone.prototype.toString = function phoneToString() {
    var ret = ` \nThis is a ${this.color} ${this.company} ${this._model}
phone` ;
    return ret;
}

// Create an instance of the Phone Class.
const m1 = new Phone("Samsung", "x11", 780, "Dark");

console.log(m1.toString())

```

`discount_price` - метод класа Phone, що змінює поле `price` при виклику

Python:

```
def discount_price(self, discount):  
    return self.price*(1-discount)
```

Js:

```
discountPrice (discount) {  
    return this.price*(1-discount);  
}
```

Реалізація Геттерів і Сеттерів.

Python:

```
def set_model(self, _model):  
    """Setter for instance variable model"""  
    match = re.search(r'^0-9', _model)  
    if(match):  
        raise ValueError('Shall be non-numeric')  
    else:  
        self._model = _model  
  
    def get_model(self):  
        """Getter for instance variable model"""  
        return self._model
```

Js:

```
set model(_model) {  
    var re = /^[^0-9]/;  
    match = _model.search(re)  
    if(_model.search(re) !== -1){  
        console.log('Shall be non-numeric')  
    }  
    else{  
        this._model = _model  
    }  
}  
  
get model() {  
    return this._model;  
}
```

Створюємо клас Smartphone. Він наслідує клас Phone, тобто створений на основі властивостей і функціоналу вже існуючого класу. Smartphone дочірній для суперкласу Phone

Python:

```
class Smartphone(Phone):

    def __init__(self, company_name, model_name, operation_system,
price, color_name):
        super().__init__(company_name, model_name, price, color_name)

        self.operation_system = operation_system

    def __str__(self):
        """Return a descriptive string for this instance, invoked by
print() and str()"""
        return f'\nThis is a {self.color} Smartphone with
{self.operation_system}'
```

Js:

```
class Smartphone extends Phone {
    constructor (company, model, price, color, operationSystem) {
        super(company, model, price, color);
        this.operationSystem = operationSystem;
    }
}
```

Створюємо клас Store

```
class Store:

    phones_amount = 5

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

    def __str__(self):
        """Return a descriptive string for this instance, invoked by
        print() and str()"""
        return f'This is a store {self.store_name}. Now available
        {self.phones_amount} phones'

    @classmethod
    def buy_phone(cls, phone):
        if Store.is_shop_open():
            cls.phones_amount -= 1
        else:
            print(f'\n\n{CRED}Shop is closed today!{CEND}')

    @classmethod
    def storage_refill(cls, amount):
        cls.phones_amount += amount

    @staticmethod
    def is_shop_open():
        week = date.today().weekday()
        return not (week == 5 or week == 6)
```

```

if __name__ == '__main__':

    store1 = Store("Test Store")
    store2 = Store("KariStore")
    print(store1)

    # @staticmethod does not have access to an instance of the class, so
    # it can be called without instantiating the class
    print (f'\n\n{CBLUE}Static method:{CEND}')

    print(store1.is_shop_open())
    print(Store.is_shop_open())

    # @classmethod bound to a class rather than an object.
    # Class methods can be called by both class and object
    print(f'\n\n{CBLUE}Class method:{CEND}')

    print(store1.phones_amount) # 5
    store2.buy_phone("phone")
    print(store1.phones_amount) # 4

    Store.storage_refill(10)
    print(store1.phones_amount) # 14
    print(f'{store1}\n{store2}')

    # delete class instance
    del store1, store2

```

@staticmethod - використовується для створення методу, який нічого не знає про клас або об'єкт класу, через який він був викликаний. Він просто отримує передані аргументи, без неявного першого аргументу, і його визначення незмінне через успадкування.

Простіше кажучи, `@staticmethod` - це ніби звичайної функції, визначеної всередині класу, яка не має доступу до примірника, тому її можна викликати без створення екземпляра класу.

`@classmethod` - це метод, який отримує клас як неявний першого аргументу, точно так же, як звичайний метод примірника отримує екземпляр. Це означає, що ви можете використовувати клас і його властивості всередині цього методу, а не конкретного екземпляра.

Простіше кажучи, `@classmethod` - це звичайний метод класу, який має доступ до всіх атрибутів класу, через який він був викликаний. Отже, `classmethod` - це метод, який прив'язаний до класу, а не до примірника класу.

Висновок: в даній роботі було розглянуто реалізацію ооп на прикладі двох, здавалось би схожих мов, із динамічною типізацією, але різними підходами до ооп.

Пайтон - об'єктно - орієнтована мова програмування, в Python об'єкти - це значення, що створюються на основі шаблону - класу . Програміст описує за допомогою спеціального синтаксису вміст класу і потім під час виконання створює об'єкти - екземпляри (instances) цього класу . У класу є свої дані - атрибути класу. До них мають доступ всі екземпляри класу. При цьому екземпляри мають свої атрибути - атрибути примірника. Ці дані доступні тільки об'єкту власнику.

JavaScript - це об'єктно-орієнтована мова, однак, в JavaScript немає концепції класу. Наприклад, об'єкт з властивостями `{name: Linda, age: 21}` не є екземпляром будь-якого класу або класу `Object` . І `Object` , і `Linda` є екземплярами самих себе. Вони визначаються безпосередньо власною поведінкою. Тут немає шару мета-даних (тобто класів), які говорили б цих об'єктів як потрібно себе вести.

Об'єкт в JavaScript - це просто колекція пар ключ-значення (і трохи магії).