Типізований Python. Модульне тестування

№ уроку: 7 **Курс:** Python Advanced

Засоби навчання: PyCharm

Огляд, мета та призначення уроку

Вивчити можливості використання типізації в Python. Отримати навички використання модуля typing. Навчитися використовувати бібліотеку «mypy» для перевірки програм, що використовують типізацію.

Отримати знання у галузі модульного тестування. Вивчити бібліотеки мови Python для тестування.

Вивчивши матеріал цього заняття, учень зможе:

- Створювати типізовані змінні та функції.
- Використовувати модуль typing
- Використовувати модуль mypy для перевірки створених типізованих модулів.
- Розуміти основні цілі модульного тестування.
- Створювати модульні тести для наявних програмних модулів.
- Використовувати бібліотеку unittest.
- Створювати заглушки для деяких ділянок коду, використовувати механізм MOCK.

Зміст уроку

1. Статична та динамічна типізація.
2. Плюси та мінуси статичної типізації.
3. Модуль typing.
4. Встановлення та використання модуля mypy.
5. Основні поняття та цілі модульного тестування.
6. Пояснення принципу створення та використання модульних тестів.
7. Приклади бібліотек для написання модульних тестів.
8. Приклад створення модульних тестів із використанням бібліотеки unittest.
9. Бібліотека pytest та приклади тестів для Django-проєкту.

Резюме

У Python 3.6 з'явився синтаксис для **анотацій** змінних (PEP 526). Суть цього PEP у тому, щоб перевести ідею анотацій типів (PEP 484) на її наступний логічний щабель, тобто уможливити зазначення типів змінних, зокрема і поля класів та об'єктів.
Варто зважити, що це нововведення не робить Python статично типізованою мовою. Інтерпретатору однаково, який тип вказаний у змінній. Проте у середовищі розробки на Python або іншому інструменті на кшталт pylint може бути активована функція перевірки анотацій, яка повідомить вам, якщо ви спочатку вказали один тип змінної, а потім спробували використовувати її як інший тип далі в програмі.
Приклад анотації
Погляньте на приклад без анотацій:

```
# untyped
value = 10
```


CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

```python
class User:
    def __init__(self, first_name, last_name):
        self.first_name = first_name
        self.last_name = last_name

def create_new_user(first_name, last_name):
    #  неоднозначність типів і перетворень
    #  first_name.???
    print(first_name)
    return User(first_name=first_name, last_name=last_name)

#  user1 = create_new_user(value, value)
user2 = create_new_user('Test1', 'Test2')
```

Додамо анотації:

```python
# typed
value: int = 10

class User:
    def __init__(self, first_name: str, last_name: str):
        self.first_name = first_name
        self.last_name = last_name

def create_new_user(first_name: str, last_name: str) -> User:
    #  автопідставлення first_name.STR_METHODS_AND_PROPS
    print(first_name.upper(), last_name.upper())
    return User(first_name=first_name, last_name=last_name)

user: User = create_new_user('Eugene', 'Test')
```

У таких анотаціях дві переваги:

1. З боку IDE програміст отримує більше підказок.
2. Програмісту легше орієнтуватися серед змінних, знаючи їхній тип.

Зверніть увагу: анотації не впливають на виконання коду в Python.
Ознайомтеся з іншими прикладами з теки **examples/typing**.

Модульне тестування

Модульне тестування необхідне для перевірки роботи модулів нашого проєкту, які наявні або розробляються. Замість ручного тестування, коли ми вручну запускаємо функції або методи класів в інтерпретаторі Python, нам необхідно написати програмний код, який тестуватиме наші модулі. Як модуль сприймаємо будь-яку програмну одиницю: функції, клас або розроблена API.

Кожен написаний тест необхідно підтримувати в актуальному стані, тобто під час зміни кодової бази необхідно запускати наявні тести та перевіряти їхню працездатність. У випадку, якщо якісь тести виконалися з помилками, необхідно перевірити коректність логіки та скоригувати або тести, або самі тестувальні модулі, – це залежить від того, де саме виявиться помилка. Тести працюють за такою схемою:

1. Генеруємо набір тестових даних для конкретного модуля/функції/класу.
2. Готуємо очікуване значення, щоб порівняти з вислідним.
3. Запускаємо виконання нашого тестувального блоку.
4. Результат виконання для кожного набору порівнюємо з відповідним вихідним набором з очікуваним результатом.
5. Якщо виникає помилка на якомусь наборі тестових даних, необхідно проаналізувати та скоригувати тестований модуль. Варто враховувати, що вибір тестових даних також має важливу роль під час тестування. Необхідно тестувати

CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine
Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

модулі не тільки на коректних вихідних даних, а щодо поведінки під час передання некоректних даних модулю.

Найчастіше виникає необхідність створювати заглушки, тобто опускати виконання будь-яких частин модуля. Наприклад, під час тестування реєстрації користувача необхідно вимкнути можливість надсилання пошти, тобто зробити заглушку для цієї функції та не навантажувати сервіс відправленням пошти під час тестування. Або як інший приклад: надсилання смс. Для цих завдань використовують механізм mock.

Наявна техніка TDD, яка розшифровується як «*test driven development*» – розробка через тестування. Ця техніка працює так: спочатку розробляється тест для покриття бажаної зміни коду або нового функціоналу, а потім – правка наявного коду модуля, приводячи його до необхідного результату. Після чого з використанням тестів проводиться рефакторинг і доопрацювання блоку, який тестується.

Наводячи у приклад фреймворк Django, він має спеціальні класи для розробки модульних тестів. Однак необхідно розуміти, що всі ці класи ґрунтуються на використанні можливостей стандартного модуля unittest. У такий спосіб ці класи просто розширюють можливості класу unittest.TestCase, додаючи допоміжні засоби для роботи з Django. Також є низка бібліотек, які дають змогу спростити написання тестів та аналіз результатів модульного тестування. Прикладом такої бібліотеки є pytest. Сама бібліотека позиціюється як бібліотека для TDD (розробка через тестування) і може використовуватися для створення тестів для того ж фреймворку Django. Ознайомтеся з іншими прикладами з теки **examples/tests**.

Закріплення матеріалу

- Що таке статична та динамічна типізація?
- Який вид типізації використовує мова Python?
- Починаючи з якої версії мова Python підтримує анотацію типів як частину синтаксису?
- Який модуль зі стандартної бібліотеки Python дає змогу комбінувати різні типи?
- Як створити тип списку з елементами типу int, використовуючи модуль typing?
- Як встановити бібліотеку mypy?
- У разі виникнення невідповідностей типів, як поводитиметься інтерпретатор Python? У чому відмінність поведінки від mypy?
- Навіщо потрібне модульне тестування?
- Опишіть основні етапи створення та виконання модульних тестів.
- Який модуль у стандартній бібліотеці Python використовується для створення модульних тестів?
- Який клас зі стандартного модуля потрібно використати, щоб створити тести?
- Який метод класу TestCase використовується під час кожного запуску окремих тестів? А який один раз для конкретного успадкованого класу?
- Що таке TDD?
- Django має власний набір інструментів для створення модульних тестів. Що їх поєднує з модулем unittest?
- Якщо виникла ситуація, в якій нам не варто виконувати якусь функцію всередині модуля (наприклад, відправлення смс користувачу), як можна обійти виконання цієї функції?

Додаткове завдання

Створіть кілька функцій:

CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

1. Обчислення середнього арифметичного списку. Якщо список порожній, то викидувати виняток **ValueError(«List is empty»**).
2. Видалення зі списку всіх значень X. Вхідні параметри: список і шукане значення для видалення всіх входжень. Функція має змінювати наявний масив, видаляючи всі входження шуканого значення.
3. Зробити функцію створення об'єкта користувача: функція приймає **first_name, last_name, birthday** і має імітувати надсилання email-повідомлення. Щоб надіслати email-повідомлення, використовуйте окрему функцію, яка друкуватиме текст повідомлення в консоль про те, що зареєстрований новий користувач. Необхідно протестувати цю функцію та реалізувати заглушку для надсилання пошти, щоб під час тестування функція не виконувала жодного відправлення пошти (друк повідомлення в консоль), а використовувалася заглушка (**mock**). Обов'язково перевірте факт виклику функції надсилання електронної пошти.

Самостійна діяльність учня

Завдання 1
Створіть функцію, яка приймає список з елементів типу int, а повертає новий список з рядкових значень вихідного масиву. Додайте анотацію типів для вхідних і вислідних значень функції.

Завдання 2
Створіть два класи Directory (тека) і File (файл) з типами (анотацією).
Клас Directory має мати такі поля:

- назва (name типу str);
- батьківська тека (root типу Directory);
- список файлів (список типу files, який складається з екземплярів File);
- список підтек (список типу sub_directories, який складається з екземплярів Directory).

Клас Directory має мати такі поля:

- додавання теки до списку підтек (add_sub_directory, який приймає екземпляр Directory та присвоює поле root для приймального екземпляра);
- видалення теки зі списку підтек (remove_sub_directory, який приймає екземпляр Directory та обнуляє поле root. Метод також видаляє теку зі списку sub_directories);
- додавання файлу в теку (add_file, який приймає екземпляр File і присвоює йому поле directory – див. клас File нижче);
- видалення файлу з теки (remove_file, який приймає екземпляр File та обнуляє у нього поле directory. Метод видаляє файл зі списку files).

Клас File має мати такі поля:

- назва (name типу str);
- тека (Directory типу Directory).

Завдання 3
Використовуючи модуль sqlite3 та модуль smtplib, реалізуйте реальне додавання користувачів до бази. Мають бути реалізовані такі функції та класи:

- клас користувача, що містить у собі такі методи: get_full_name (ПІБ з поділом через пробіл: «Петров Ігор Сергійович»), get_short_name (формату ПІБ: «Петров І. С.»), get_age (повертає вік користувача, використовуючи поле birthday типу datetime.date); метод __str__ (повертає ПІБ та дату народження);
- функція реєстрації нового користувача (приймаємо екземпляр нового користувача та відправляємо email на пошту користувача з листом подяки).

CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine
Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

- функція відправлення email з листом подяки.
- функція пошуку користувачів у таблиці users за іменем, прізвищем і поштою.

Протестувати цей функціонал, використовуючи заглушки у місцях надсилання пошти. Під час штатного запуску програми вона має відправляти повідомлення на вашу реальну поштову скриньку (необхідно налаштувати SMTP, використовуючи доступи від провайдера вашого email-сервісу).

Приклад налаштування SMTP для сервісу Gmail:
https://support.google.com/mail/answer/7126229?hl=ru

Рекомендовані ресурси

Офіційний сайт Python (3.6) – typing
https://docs.python.org/3.6/library/typing.html
Офіційний сайт пакета mypy
https://mypy.readthedocs.io/en/latest/
Офіційний сайт Python – Metaclasses
https://docs.python.org/3.6/library/unittest.html
Офіційний сайт Python – Mock objects
https://docs.python.org/3/library/unittest.mock.html
Офіційний сайт документації pytest
https://docs.pytest.org/en/latest/

CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com