Типизированный Python. Модульное тестирование

№ урока: 7 **Курс:** Python Advanced

Средства обучения: PyCharm

Обзор, цель и назначение урока

Изучить возможности использования типизации в Python. Получить навыки использования модуля typing. Использовать библиотеку «mypy» для проверки программ, использующих типизацию.
Получение знаний в области модульного тестирования. Изучение библиотек языка Python для задач тестирования.

Изучив материал данного занятия, учащийся сможет:

- Создавать типизированные переменные и функции
- Использовать модуль typing
- Использовать модуль mypy для проверки созданных типизированных модулей
- Понимать основные цели модульного тестирования.
- Создавать модульные тесты для уже имеющихся программных модулей.
- Умение использовать библиотеку unittest.
- Создавать заглушки для некоторых участков кода, использовать механизм mock.

Содержание урока

1. Статическая и динамическая типизация.
2. Плюсы и минусы статической типизации.
3. Модуль typing.
4. Установка и использование модуля mypy
5. Основные понятия и цели модульного тестирования.
6. Объяснение принципа создания и использования модульных тестов.
7. Примеры библиотек для написание модульных тестов.
8. Пример создания модульных тестов с использованием библиотеки unittest.
9. Библиотека pytest и примеры тестов для Django проекта.

Резюме

В Python 3.6 появился синтаксис для **аннотаций** переменных (PEP 526). Суть этого PEP заключается в том, чтобы перевести идею аннотаций типов (PEP 484) на её следующую логическую ступень, т.е. сделать возможным указание типов переменных, включая поля классов и объектов.
Следует принять во внимание, что это нововведение не делает Python статически типизированным языком. Интерпретатору всё равно какой тип указан у переменной. Тем не менее в среде разработки на Python или другом инструменте вроде pylint может быть включена функция проверки аннотаций, которая сообщит вам, если вы сначала указали один тип переменной, а затем попытались использовать её в качестве другого типа далее в программе.

Пример аннотации

Взгляните на пример без аннотаций:

```
# 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):
    # неоднозначность типов и преобразований
    # 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')
```

Добавим аннотации:

```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:
    # автоподстановка 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')
```

В таких аннотациях два преимущества:

1. Со стороны IDE программист получает больше подсказок
2. Программисту легче ориентироваться среди переменных, зная их тип

Обратите внимание, что аннотации не повлияют на выполнение кода в Python.
Ознакомитесь с другими примерами из папки ***examples/typing.***

Модульное тестирование

Модульное тестирование необходимо для проверки работы имеющихся или разрабатываемых модулей нашего проекта. Вместо ручного тестирования, когда мы вручную запускаем функции или методы классов в интерпретаторе Python, нам необходимо написать программный код, который будет тестировать наши модули. В качестве модуля понимается любая программная единица, будь то функции, класс или разработанная API.
Каждый написанный тест необходимо поддерживать в актуальном состоянии, то есть при изменении кодовой базы необходимо запускать имеющиеся тесты и проверять их работоспособность. В случае, если какие-либо тесты выполнились с ошибками, необходимо проверить корректность логики и скорректировать либо тесты, либо сами тестируемые модули - это зависит от того, где именно обнаружится ошибка.
Тесты работают по следующей схеме:

1. Генерируем набор тестовых данных для конкретного модуля/функции/класс.
2. Подготавливаем ожидаемые значение, чтобы сравнить с результирующими.
3. Запускаем выполнение нашего тестируемого блока на данных наборах.
4. Результат выполнения для каждого набора сравниваем с соответствующим исходному набору с ожидаемым результатом.
5. Если возникает ошибка на каком-либо наборе тестовых данных, то необходимо проанализировать и скорректировать тестируемый модуль. Следует учитывать, что выбор тестовых данных тоже имеет важную роль при тестировании. Необходимо тестировать

CyberBionic systematics

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

модули не только на корректных исходных данных, но и на предмет поведения при передачах модулю некорректных данных.

Зачастую возникает необходимость создавать заглушки, то есть опускать выполнение каких-либо частей модуля. Например, при тестировании регистрации пользователя необходимо отключить возможность отправки почты, то есть сделать заглушку для данной функции и не нагружать сервис отправкой почты во время тестирования. Или как другой пример: отправка смс. Для данных задач используется механизм mock.

Существует техника TDD, которая расшифровывается как "*test driven development*" - Разработка через тестирование. Данная техника следует следующим принципам: сначала разрабатывается тест для покрытия желаемого изменения кода или нового функционала, а затем следует правка имеющегося кода модуля, приводя его к требуемому результату. После чего с использованием тестов проводится рефакторинг и доработка тестируемого блока.

Приводя в пример фреймворк Django, он имеет специальные классы, для разработки модульных тестов. Однако, необходимо понимать, что все эти классы основаны на использовании возможностей стандартного модуля unittest. Таким образом, данные классы просто расширяют возможности unittest.TestCase класса, добавляя вспомогательные средства для работы с Django. Также, существует ряд библиотек, которые позволяют упростить написание тестов и анализ результатов модульного тестирования. Примером такой библиотеки является pytest. Сама библиотека позиционируется как библиотека для TDD (Разработка через тестирование) и может использоваться для создания тестов для того же фреймворка Django.

Ознакомитесь с другими примерами из папки ***examples/tests.***

Закрепление материала

- Что такое статическая и динамическая типизация?
- Какой вид типизации использует язык Python?
- Начиная с какой версии язык Python поддерживает аннотацию типов, как часть синтаксиса?
- Какой модуль из стандартной библиотеки Python позволяет комбинировать различные типы?
- Как создать тип списка с элементами типа int используя модуль typing?
- Как установить библиотеку mypy?
- В случае возникновения несоответствий типов, как будет вести себя интерпретатор Python и в чем отличие поведения от mypy?
- Зачем нужно модульное тестирование?
- Опишите основные этапы создания и выполнения модульных тестов?
- Какой модуль в стандартной библиотеке языка Python используется для создания модульных тестов?
- Какой класс из стандартного модуля нужно использовать, чтобы создать тесты?
- Какой метод класса TestCase используется при каждом запуске отдельных тестов и какой только один раз для конкретного класса унаследованного?
- Что такое TDD?
- Django имеет свой набор собственный набор инструментов для создания модульных тестов, что их объединяет с модулем unittest?
- В случае, если возникла ситуация, в которой нам не следует выполнять какую-то функцию внутри тестируемого модуля (например, отправку смс пользователю), как можно обойти выполнение данной функции?

Дополнительное задание

Задание

Создайте несколько функций:

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. Вычисление среднего арифметического списка. В случае, если список пустой, то выбрасывать исключение **ValueError(«List is empty**»);
2. Удаление из списка всех значений X. Входные параметры: список и искомое значение для удаления всех вхождений. Функция должна изменять имеющийся массив, удаляя все вхождения искомого значения.
3. Сделать функция создания объекта пользователя: функция принимает **first_name**, **last_name**, **birthday** и должна имитировать отправку Email-сообщения. Для отправки Email-сообщения используйте отдельную функцию, которая по факту будет печатать текст сообщения в консоль, о том, что зарегистрирован новый пользователь. Необходимо протестировать данную функцию и реализовать заглушку для отправки почты, чтобы во время тестирования функция не выполняла никакой отправки почты (печать сообщения в консоль), а использовалась заглушка (**mock**). Обязательно проверить факт вызова функции отправки Email.

Самостоятельная деятельность учащегося

Задание 1
Создать функцию, которая принимает список из элементов типа int, а возвращает новый список из строковых значений исходного массива. Добавить аннотацию типов для входных и результирующих значений функции.

Задание 2
Создайте два класса Directory (папка) и File (файл) с типами (аннотацией).
Класс Directory должен иметь следующие поля:

- Название (name типа str);
- Родительская папка (root типа Directory);
- Список файлов (files типа список, состоящий из экземпляров File)
- Список подпапок (*sub_directories* типа список, состоящий из экземпляров Directory).

Класс Directory должен иметь следующие поля:

- Добавление папки в список подпапок (add_sub_directory принимающий экземпляр Directory и присваивающий поле root для принимаемого экземпляра);
- Удаление папки из списка подпапок (remove_sub_directory, принимающий экземпляр Directory и обнуляющий у него поле root. Метод также удаляет папку из списка sub_directories).
- Добавление файла в папку (add_file, принимающий экземпляр File и присваивающий ему поле directory - см. класс File ниже);
- Удаление файла из папки (remove_file, принимающий экземпляр File и обнуляющий у него поле directory. Метод удаляет файл из списка files).

Класс File должен иметь следующие поля:

- Название (name типа str);
- Папка (directory типа Directory);

Задание 3
Используя модуль **sqlite3** и модуль **smtplib**, реализуйте реальное добавление пользователей в базу. Должны быть реализовать следующие функции и классы:
- класс пользователя, содержащий в себе следующие методы: **get_full_name** (ФИО с разделением через пробел: «Петров Игорь Сергеевич»), **get_short_name** (ФИО формата: «Петров И. С.»), **get_age** (возвращает возраст пользователя, используя поле **birthday** типа **datetime.date**); метод **__str__** (возвращает ФИО и дату рождения).
- функция регистрации нового пользователя (принимаем экземпляр нового пользователя и отправляем Email на почту пользователя с благодарственным письмом).

CyberBionic systematics
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

- функция отправки Email с благодарственным письмом.
- функция поиска пользователей в таблице **users** по имени, фамилии и почте.

Протестировать данный функционал, используя заглушки, в местах отправки почты. При штатном запуске программы, она должна отправлять сообщение на ваш реальный почтовый ящик (необходимо настроить SMTP, используя доступы от провайдера вашего Email-сервиса).
Пример настройки SMTP для сервиса Gmail: https://support.google.com/mail/answer/7126229?hl=ru

Рекомендуемые ресурсы

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

CyberBionic systematics
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