Модульное тестирование

№ урока: 7 **Курс:** Python Advanced

Средства обучения: Python3.6, PyCharm

Обзор, цель и назначение урока

Получение знаний в области модульного тестирования. Изучение библиотек языка Python для задач тестирования.

Изучив материал данного занятия, учащийся сможет:

- 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. Основные понятия и цели модульного тестирования.
2. Объяснение принципа создания и использования модульных тестов.
3. Примеры библиотек для написание модульных тестов.
4. Пример создания модульных тестов с использованием библиотеки unittest.
5. Библиотека pytest и примеры тестов для Django проекта.

Резюме

- 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 любом 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:
    - Генерируем набор тестовых данных для конкретного модуля/функции/класс.
    - Подготавливаем ожидаемые значение, чтобы сравнить с результирующими.
    - Запускаем выполнение нашего тестируемого блока на данных наборах.
    - Результат выполнения для каждого набора сравниваем с соответствующим исходному набору с ожидаемым результатом.
    - 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 модули не только на корректных исходных данных, но и на предмет поведения при передачи модулю некорректных данных.

CyberBionic systematics

CyberBionic Systematics ® 2019
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 750 312
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

- 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.

Закрепление материала

- 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:

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

CyberBionic systematics
CyberBionic Systematics ® 2019
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine
Tel. 0 800 750 312
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

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
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).
- 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 SMTP для сервиса Yandex: https://yandex.ru/support/mail/mail-clients.html#imap

Рекомендуемые ресурсы

Официальный сайт Python (3.6) - Metaclasses
https://docs.python.org/3.6/library/unittest.html
Официальный сайт Python (2.7) – 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 ® 2019
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine
Tel. 0 800 750 312
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com