Библиотеки Python (часть 2)

Для тестирования

Для проверки работоспособности программ обычно устраивают тестирование. Это отдельное направление в IT. Для кода на Python также пишут тесты, генерируют для них данные, меняют настройки среды и многое другое. Для упрощения таких задач тоже существуют специальные библиотеки.

Responses. Ранее мы говорили про requests — это ее противоположность. Responses помогает генерировать различные ответы на запросы и затем анализировать их результаты. В отличие от requests, responses используется в основном при тестировании — помогает проверять, как приложение реагирует на разные ответы внешнего сервиса.

```
import responses
import requests
@responses.activate
def test_my_api():
    responses.add(responses.GET, 'http://twitter.com/api/1/foobar',
                  body='{"error": "not found"}', status=404,
                  content_type='application/json')

    resp = requests.get('http://twitter.com/api/1/foobar')

    assert resp.json() == {"error": "not found"}

    assert len(responses.calls) == 1
    assert responses.calls[0].request.url ==
'http://twitter.com/api/1/foobar'
    assert responses.calls[0].response.text == '{"error": "not found"}'
```

Возврат строки в ответе

Freezegun. Библиотека поможет, если нужно протестировать работу программы в конкретный период. Она «замораживает» параметры даты и времени на определенной точке, нужной программисту. Это может быть полезно, например, если тестируемая программа должна работать по-разному в зависимости от времени суток или дня недели.

Freezegun предоставляет freeze_time() декоратор, который мы можем использовать для установки фиксированной даты для наших тестовых функций. Получение из последнего раздела, это помогает развивать это:

```python
fake_date = Mock(wraps=datetime.date)
fake_date.today.return_value = datetime.date(2022, 11, 10)

@patch('tomorrow.date', fake_date)
def test_tomorrow():
   assert tomorrow.tomorrow() == datetime.date(2022,11, 11)



from freezegun import freeze_time

@freeze_time('2020-07-02')
def test_tomorrow():
   assert tomorrow.tomorrow(include_time=True) == datetime.datetime(2022, 11,11)
```

Faker. Для тестирования функциональности приложения часто нужно заполнять его данными, например о пользователях. Использовать реальные персональные данные в такой ситуации некорректно и небезопасно. Поэтому лучше воспользоваться Faker — библиотекой, которая помогает сгенерировать большое количество ненастоящей информации. Это могут быть ФИО, телефоны, даты рождения, данные карточек и многое другое.

Для начала, необходимо создать и инициализировать faker генератор.

```python
from faker import Faker
fake = Faker()
```

Инициализация faker генератора

Для того, чтобы faker сгенерировал данные, нужно обратиться к соответствующему свойству. Например, такой код

```python
for i in range(1, 4):
    print(i, fake_ru.name())
```

Генерируем имена

может сгенерировать следующие данные:

1. Robin Davis
2. Brittany Cooper
3. Matthew Quinn

faker.providers.misc свойство *password*.

1. &NvTrwCMn1qkD1_C
2. z0N7_PXe^ckaSXTM
3. !q5G6+Mj0S$2o0Yh

```
fake.password(length=16, special_chars=True, digits=True, upper_case=True, lower_case=True)
```

faker.providers.phone_number свойство *phone_number*

1. +7 371 063 0825
2. +7 (027) 657-18-36
3. 85326556498

```
fake_ru.phone_number()
```

Factory_boy. Еще один инструмент для генерирования данных. В отличие от Faker, он генерирует фикстуры — блоки кода, которые выполняются до или после тестовых функций. Фикстуры помогают привести приложение в нужное состояние, задать исходные данные или сгенерировать информацию для теста. Эта библиотека упрощает их создание и применяется в основном в unit-тестировании.

Чтобы разобраться с понятием юнит-тестирования, надо для начала узнать определение юнитов. Любая программа состоит из юнитов, или модулей, — отдельных блоков и функций. Кнопка добавления товара в корзину, формула калькулятора стоимости, скрипт для формирования карточки товара — всё это отдельные модули.

Что такое модульное тестирование

Модули взаимодействуют и обеспечивают работу программы или приложения. Чтобы проверить, правильно ли написан модуль, проводят юнит-тесты, или модульное тестирование, — проверку не всего приложения, а одного модуля. Пример юнит-теста — проверка функции подсчёта общей стоимости заказа.

Модульное тестирование проводят сразу после написания кода. Проверить работу кнопки в готовом приложении не получится, потому что на неё уже влияют другие модули.

Для машинного обучения и ИИ

Машинное обучение, анализ данных, искусственный интеллект — это области, в которых Python используется как основной язык. Он применяется в распознавании, предсказании, генерации данных, в том числе в компьютерном зрении, обработке естественного языка и других направлениях. Библиотеки, как всегда, упрощают работу с задачами. Также они используются для математических расчетов, построения графиков и пр.

NumPy. Предназначена для работы с числами и сложной математикой. В первую очередь она облегчает расчеты с матрицами и многомерными массивами — именно в таком виде мы передаем любые данные на вход алгоритмам и моделям в методах глубокого обучения. Поэтому NumPy входит в базовый стек библиотек для Machine Learning.

SciPy. Основана на NumPy, но имеет более широкий функционал. Она предназначена для глубоких и сложных математических операций и вычислений. В SciPy много функций для научного анализа и работы с высшей математикой.

Matplotlib. Используется для удобного построения графиков и визуализации результатов. Активно применяется в задачах анализа данных, при оценке и сравнении метрик алгоритмов, наблюдениях за моделью. Нередко Matplotlib используется в тандеме с NumPy и SciPy.

Pillow. Предназначена для работы с картинками. Библиотека помогает их обрабатывать, применять различные эффекты, «чистить» от шума, работать с пикселями и делать многое другое. Обработка изображений обычно используется в задачах компьютерного зрения, когда нужно перевести картинку в понятный для компьютера формат.

TensorFlow. Мощная библиотека для глубокого обучения. В основном TensorFlow используется для создания и обучения нейронных сетей. Ее можно представить как «ядро» для математических вычислений на Python. Она представляет данные как тензоры — векторы, которые складываются в графы.

Keras. Упрощает использование TensorFlow. Keras отвечает за создание и настройку моделей и нейросетей, а TensorFlow выполняет в них расчеты.

PyTorch. Еще одна библиотека с открытым исходным кодом от Facebook*, которая используется для глубокого обучения, создания и обучения нейронных сетей. Она более новая, чем TensorFlow и показывает по сравнению с ней лучшие результаты.

Pymorphy2. Это морфологический анализатор русского языка. Он распознает и приводит слова к нормальной форме (например, «люди -> человек», или «гулял -> гулять»), меняет число, род, падеж и пр.

Pymystem3. Библиотека от Яндекса имеет точно такой же функционал, как и Pymorphy2, и является ее альтернативой. Pymystem3 быстро и качественно распознает части речи и лексемы слова.

OpenCV. Это открытая библиотека для работы с алгоритмами компьютерного зрения, машинным обучением и обработкой изображений.