Приемы промышленной разработки

Аксентьев Артем Алексеевич

Chillers

31 марта 2020 г.

О чем пойдет речь...

1. Пакеты в Python
2. Виртуальные окружения
   - virtualenv
   - venv
3. Документирование кода

Пакеты в Python

Классификация пакетов Python

1. Стандартные пакеты;

Пакеты в Python

Классификация пакетов Python

1. Стандартные пакеты;
2. Пакеты из PyPI;

Пакеты в Python

Классификация пакетов Python

1. Стандартные пакеты;
2. Пакеты из PyPI;
3. Сторонние пакеты.

Пакеты в Python

Классификация пакетов Python

1. Стандартные пакеты;
2. Пакеты из PyPI;
3. Сторонние пакеты.

PyPI

PyPI – это Python Package Index (PyPI) – репозиторий пакетов Python, доступный для любого разработчика и пользователя Python.

Установка пакетов

Как устанавливать пакеты?

1. С помощью менеджера пакетов;
   - `pip <аргументы>`;
   - `python -m pip <аргументы>`.
2. Вручную;
   1. Скачать исходный код;
   2. Выполнить код из файла setup.py.

Установка pip

Как устанавливить pip?

1. Скачать файл get-pip с `https://bootstrap.pypa.io/get-pip.py`;
2. Выполнить этот файл с помощью python.

```
> curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py
> python get-pip.py
```

Установка с помощью pip

Установка последней версии пакета

```
$ pip install ProjectName
```

Установка определенной версии пакета

```
$ pip install ProjectName==3.2
```

Удаление пакета

```
$ pip uninstall ProjectName
```

Обновление пакетов

```
$ pip install --upgrade ProjectName
```

Альтернативный способ использовать pip

```
$ python3.7 -m pip install ProjectName
```

Проблемы при разработке на Python

Зачем использовать виртуальные окружения?

1. Разные программы могут требовать разные версии библиотек;

Проблемы при разработке на Python

Зачем использовать виртуальные окружения?

1. Разные программы могут требовать разные версии библиотек;
2. Хочется быть уверенным, что программа будет работать именно с этой версией библиотеки;

Проблемы при разработке на Python

Зачем использовать виртуальные окружения?

1. Разные программы могут требовать разные версии библиотек;
2. Хочется быть уверенным, что программа будет работать именно с этой версией библиотеки;
3. Может не быть доступа к глобальным библиотекам;

Проблемы при разработке на Python

Зачем использовать виртуальные окружения?

1. Разные программы могут требовать разные версии библиотек;
2. Хочется быть уверенным, что программа будет работать именно с этой версией библиотеки;
3. Может не быть доступа к глобальным библиотекам;
4. Быстрая «развертка» приложения.

Инструменты для решения этой проблемы

Стандартные инструменты

1. venv - только Python 3.

Инструменты для решения этой проблемы

Стандартные инструменты

1. venv - только Python 3.

Иснтрументы из PyPI

1. virtualenv - один из самых популярных методов создания виртуального окружения.

Инструменты для решения этой проблемы

Стандартные инструменты

1. venv - только Python 3.

Иснтрументы из PyPI

1. virtualenv - один из самых популярных методов создания виртуального окружения.
2. pyenv - инструмент для изоляции версий Python

Инструменты для решения этой проблемы

Стандартные инструменты

1. venv - только Python 3.

Иснтрументы из PyPI

1. virtualenv - один из самых популярных методов создания виртуального окружения.
2. pyenv - инструмент для изоляции версий Python
3. virtualenvwrapper - обертка для virtualenv позволяющая хранить все изолированные окружения в одном месте, создавать их, копировать и удалять.

Создание виртуального окружение virtualenv

Установка virtualenv

```
> pip install virtualenv
```

Создание виртуального окружния

```
> virtualenv envName
```

Структура каталога

Структура директории

```
> tree -L 2
.
|--- bin # Файлы для взаимодействия с виртуальной средой
| |--- activate
| |--- activate.csh
| |--- activate.fish
| |--- activate.ps1
| |--- activate.xsh
| |--- activate_this.py
....
|--- include # C-заголовки
|--- lib # копия версии Python вместе с «site-packages»
| |---python3.7
```

Работа с виртуальной средой

Активация виртуальной среды

```
> source env/bin/activate
```

Деактивация виртуальной среды

```
(env) > deactivate
```

Создание виртуального окружение venv

Создание виртуального окружния

```
> python -m venv envName_2
```

Активация виртуальной среды

```
> source envName_2/bin/activate
```

Деактивация виртуальной среды

```
(envName_2) > deactivate
```

Зачем писать комментарии?

Функции комментариев

1. Улучшают навигацию. Выделение мест в коде.
2. Объясняют написанное. Сложные алгоритмические трюки или формулы.
3. Документировать код.
4. Описывают результаты.

Важно

Комментарий должен дополнять код, а не заменять его.

Как писать комментарии?

Чистый код

1. “Код — сам себе лучшая документация”.
2. Комментарий должен отвечать на вопрос “Почему?”, а не “Как?”.
3. Комментарии не должны устаревать.
4. Пишите документационные комментарии.

Строки документации

Определение

Строки документации - строковые литералы, которые являются первым оператором в модуле, функции, классе или определении метода. Такая строка документации становится специальным атрибутом `__doc__` этого объекта.

Строки документации

Однострочная строка документации

```
def function(a, b):
    """Do X and return a list."""
```

Многострочная строка документации

```
def complex(real=0.0, imag=0.0):
    """Form a complex number.

    Keyword arguments:
    real -- the real part (default 0.0)
    imag -- the imaginary part (default 0.0)

    """
    if imag == 0.0 and real == 0.0: return complex_zero
```

Как документировать?

Строка документации программы

“Сообщения по использованию”. Максимально полно описывает взаимодействие с Вашей программой.

Строка документации модуля

Перечисление классов, исключений, функций (и любых других объектов), которые экспортируются модулем, с краткими пояснениями (в одну строчку) каждого из них.

Строка документации функции

Обобщают поведение функции или метода и документирует свои аргументы, возвращаемые значения, побочные эффекты, исключения, дополнительные аргументы, именованные аргументы, и ограничения на вызов функции.

Системы автоматической генерации документации

Зачем?

Преобразовывать комментарии в коде в документацию.

Какие бывают?

1. Doxygen:
   - Стандарт документирования в области C/C++;
   - Быстрое начало работы;
   - Перегруженные комментарии.

Системы автоматической генерации документации

Зачем?

Преобразовывать комментарии в коде в документацию.

Какие бывают?

1. Doxygen:
   - Стандарт документирования в области C/C++;
   - Быстрое начало работы;
   - Перегруженные комментарии.
2. Sphinx:
   - С недавнего времени стандарт документации в Python;
   - Требуется долгая настройка;
   - Sphinx использует синтаксис разметки текста reStructuredText;
   - Можно писать более приятные комментарии.

Примеры документирования

Doxygen

```
1  """!@brief Модуль с реализацией оконного интерфейса программы.
2
3  @author Don Sangre
4  """
5  import sys
6  import json
7
8
9  class Item(QtGui.QStandardItem):
10     """!@brief Класс содержимого для QListView.
11
12     @param name --  название сайта
13     @param url -- адрес сайта
14     """
15
16     def __init__(self, name, url):
17         """!@brief Инициализатор.
18
19         @param name --  название сайта
20         @param url -- адрес сайта
21         """
22         super(Item, self).__init__(name)
23         self._url = url
```

Примеры документирования

Sphinx

```
"""
Модуль с константами.

Attributes:
    N (int): количество строк поля (:class:`.Field`).
    M (int): количество столбцов поля (:class:`.Field`).
    THRESHOLD (int): порог для ограничения количества действий робота
        (:class:`.Robot`).
    NUM_BARRIERS (int): количество препятствий на карте (:class:`.Field`).
"""

# Количество строк
N = 7
# Количество столбцов
M = 6
# Ограничение по количеству действий робота
THRESHOLD = N * M * 10
# Количество препятствий на карте
NUM_BARRIERS = 14
```

Литературные источники I

1. pip 20.0.2 user guide
2. Документация Virtualenv
3. Документация Venv
4. PEP 405 – Python Virtual Environments
5. Памятка по virtualenv
6. Программирование на Python. Виртуальное окружение и пакеты.
7. PEP 8 – Style Guide for Python Code
8. PEP 257 – Docstring Conventions
9. PEP 8 – Style Guide for Python Code
10. Документация Doxygen
11. Документация Sphinx
12. Роберт Мартин. Чистый код. Создание, анализ и рефакторинг.

Спасибо!