KOLT Python

Python Modules and Third-Party Packages

Halil Eralp Koçaş

Tuesday 7th April, 2020

KOÇ UNIVERSITY
OFFICE OF LEARNING AND TEACHING

Agenda

1. Python Modules and Packages

2. Package Management

3. Virtual Environments

Python Modules

- Modules are files that contains Python statements and definitions

Python Modules

- Modules are files that contains Python statements and definitions
- Modules help you to break down large programs into small manageable and organized files

Python Modules

- Modules are files that contains Python statements and definitions
- Modules help you to break down large programs into small manageable and organized files
- Modules provides reusability of a code

Python Modules

- Modules are files that contains Python statements and definitions
- Modules help you to break down large programs into small manageable and organized files
- Modules provides reusability of a code
- Most used functions can be defined in Modules and can be used in other programs without copying the definitions into every program

Python Modules

- Modules are files that contains Python statements and definitions
- Modules help you to break down large programs into small manageable and organized files
- Modules provides reusability of a code
- Most used functions can be defined in Modules and can be used in other programs without copying the definitions into every program
- To use definitions in modules, modules are needed to be **imported**

Python Packages

- Packages can be thought as directories in which multiple modules are present

Python Packages

- Packages can be thought as directories in which multiple modules are present
- Packages are organized hierarchically

Python Packages

- Packages can be thought as directories in which multiple modules are present
- Packages are organized hierarchically
- Packages can contain subpackages as well as regular modules

Python Packages

- Packages can be thought as directories in which multiple modules are present
- Packages are organized hierarchically
- Packages can contain subpackages as well as regular modules
- All packages are modules but not all modules are packages

Importing Modules

Importing Modules

Import a module:

Importing Modules

Import a module:

```
import module_name # all definitions
```

Importing Modules

Import a module:

```
import module_name # all definitions
import module_name as name # module can be used with "name"
```

Importing Modules

Import a module:

`import module_name` # all definitions

`import module_name as name` # module can be used with ”name”

`from module_name import func1` # only specified names, func1

Importing Modules

Import a module:

`import module_name` # all definitions

`import module_name as name` # module can be used with ”name”

`from module_name import func1` # only specified names, func1

`from module_name import func1 as function` # func1 will be used by calling ”function”

Importing Modules

Import a module:

`import module_name` # all definitions

`import module_name as name` # module can be used with ”name”

`from module_name import func1` # only specified names, func1

`from module_name import func1 as function` # func1 will be used by calling ”function”

`from module_name import *` # all names in module

Python Package Index (PyPI)

Python Package Index (PyPI)

Repository of software for the Python programming language.

Python Package Index (PyPI)

Repository of software for the Python programming language.

- 23,000+ Python3 packages.

Python Package Index (PyPI)

Repository of software for the Python programming language.

- 23,000+ Python3 packages.
- If you want a package, PyPI probably has it.

Python Package Index (PyPI)

Repository of software for the Python programming language.

- 23,000+ Python3 packages.
- If you want a package, PyPI probably has it.

Visit *pypi.org* to explore packages.

pip

pip

- Recommended tool for installing Python packages.

pip

- Recommended tool for installing Python packages.
- `pip` is already installed with modern Python distributions.

pip

- Recommended tool for installing Python packages.
- **`pip`** is already installed with modern Python distributions.
- Try `pip -V` on your command line/terminal(`pip3 -V` for Macs).

pip

- Recommended tool for installing Python packages.
- **`pip`** is already installed with modern Python distributions.
- Try `pip -V` on your command line/terminal(`pip3 -V` for Macs).

```
$ pip -V
pip 20.0.2 from --PATH_TO_PIP-- (python 3.5)
```

pip

- Recommended tool for installing Python packages.
- **`pip`** is already installed with modern Python distributions.
- Try `pip -V` on your command line/terminal(`pip3 -V` for Macs).

```
$ pip -V
pip 20.0.2 from --PATH_TO_PIP-- (python 3.5)
$ python -m pip -V
pip 20.0.2 from --PATH_TO_PIP-- (python
version)
```

Common pip commands

Common pip commands

Install a package:

Common pip commands

Install a package:

```
$ pip install package_name # latest version
```

Common pip commands

Install a package:

```
$ pip install package_name # latest version
$ pip install package_name==1.0.1 # specific version
```

Common pip commands

Install a package:

```
$ pip install package_name # latest version
$ pip install package_name==1.0.1 # specific version
$ pip install package_name>=1.0.1 # minimum version
```

Common pip commands

Install a package:

```
$ pip install package_name # latest version
$ pip install package_name==1.0.1 # specific version
$ pip install package_name>=1.0.1 # minimum version
```

Uninstall a package:

Common pip commands

Install a package:

```
$ pip install package_name # latest version
$ pip install package_name==1.0.1 # specific version
$ pip install package_name>=1.0.1 # minimum version
```

Uninstall a package:

```
$ pip uninstall package_name
```

Common pip commands

Install a package:

```
$ pip install package_name # latest version
$ pip install package_name==1.0.1 # specific version
$ pip install package_name>=1.0.1 # minimum version
```

Uninstall a package:

```
$ pip uninstall package_name
```

Update a package:

Common pip commands

Install a package:

```
$ pip install package_name # latest version
$ pip install package_name==1.0.1 # specific version
$ pip install package_name>=1.0.1 # minimum version
```

Uninstall a package:

```
$ pip uninstall package_name
```

Update a package:

```
$ pip install --upgrade package_name
```

Common pip commands

Install a package:

```
$ pip install package_name # latest version
$ pip install package_name==1.0.1 # specific version
$ pip install package_name>=1.0.1 # minimum version
```

Uninstall a package:

```
$ pip uninstall package_name
```

Update a package:

```
$ pip install --upgrade package_name
```

Search PyPI for matches:

Common pip commands

Install a package:

```
$ pip install package_name # latest version
$ pip install package_name==1.0.1 # specific version
$ pip install package_name>=1.0.1 # minimum version
```

Uninstall a package:

```
$ pip uninstall package_name
```

Update a package:

```
$ pip install --upgrade package_name
```

Search PyPI for matches:

```
$ pip search query
```

Virtual Environments

Virtual Environments

A *virtual environment* is an isolated Python environment that contains the Python interpreter, installed libraries and scripts.

Virtual Environments

A *virtual environment* is an isolated Python environment that contains the Python interpreter, installed libraries and scripts.
Why do we need them?

Virtual Environments

A *virtual environment* is an **isolated** Python environment that contains the Python interpreter, installed **libraries** and scripts.
Why do we need them?

Virtual Environments

A *virtual environment* is an **isolated** Python environment that contains the Python interpreter, installed **libraries** and scripts.
Why do we need them?
What happens if two different programs use the same library?

Virtual Environments

A *virtual environment* is an **isolated** Python environment that contains the Python interpreter, installed **libraries** and scripts.
Why do we need them?
What happens if two different programs use the same library?

- We might want to use different versions of the same library.

Virtual Environments

A *virtual environment* is an **isolated** Python environment that contains the Python interpreter, installed **libraries** and scripts.
Why do we need them?
What happens if two different programs use the same library?

- We might want to use different versions of the same library.
- Updating a library for **Program A** can harm another **Program B**. (*Breaking Changes*)

Virtual Environments

A *virtual environment* is an **isolated** Python environment that contains the Python interpreter, installed **libraries** and scripts.
Why do we need them?
What happens if two different programs use the same library?

- We might want to use different versions of the same library.
- Updating a library for **Program A** can harm another **Program B**. (*Breaking Changes*)
- We want **isolation** between programs.

Creating Virtual Environments

Creating Virtual Environments

In Python 3.6+, the recommended way to create a virtual environment is using **venv** package, which is included in the standard installation (similar to **pip**).

Creating Virtual Environments

In Python 3.6+, the recommended way to create a virtual environment is using **venv** package, which is included in the standard installation (similar to **pip**).

Creating a virtual environment:

```
$ python -m venv /path/to/new/virtualenv
```

Creating Virtual Environments

In Python 3.6+, the recommended way to create a virtual environment is using **venv** package, which is included in the standard installation (similar to **pip**).

Creating a virtual environment:

```
$ python -m venv /path/to/new/virtualenv
```

Activating a virtual environment:

cd(Change directory) to virtual environment folder.

In Windows: `$ Scripts\activate`

In Mac/Linux: `$ source bin/activate`

Creating Virtual Environments

In Python 3.6+, the recommended way to create a virtual environment is using **venv** package, which is included in the standard installation (similar to **pip**).

Creating a virtual environment:

```
$ python -m venv /path/to/new/virtualenv
```

Activating a virtual environment:

cd(Change directory) to virtual environment folder.

In Windows: `$ Scripts\activate`

In Mac/Linux: `$ source bin/activate`

Deactivating a virtual environment:

```
$ deactivate
```