PYTHON PACKAGES

CHRISTOPH PAULIK

WHAT ARE PACKAGES

PACKAGES AND MODULES

Module

.py file that contains functions/classes or variables

Package

Collection of python modules. Can be nested in folders

FOLDERS ON FILESYSTEM

- my_package
  - __init__.py
  - my_module.py

__init__.py tells python that the folder contains modules

code in init.py is executed during import!

http://docs.python.org/2/tutorial/modules.html

USING PACKAGES

- Packages have to be imported for the program to be able to use them

```
import math # import complete module
import very_long_package_name as vlp # shorten the name
from datetime import date # import only certain variables/classes/functions
```

  - Avoid `from package import *` -> dirty namespace

Example - Open `WinPython Command Prompt.exe` or `Shell`

WHERE DO I LOOK FOR PACKAGES?

- Standard library
- Python Package Index (PyPI) (56338 packages last week)
- Search Engine „python <your problem here>“
  - Project websites
  - Stackoverflow
  - Blogs

IMPORTANT STANDARD LIBRARY PACKAGES

DATETIME MODULE

Representation of dates and times. Documentation.

```
from datetime import date
a = date(2015, 3, 1)
b = date.today()
print(a, b)
print(a.year, a.month, a.day) # attribute access
delta = b - a # difference are a datetime.timedelta object
print(delta)
print(type(delta))
```

```
(datetime.date(2015, 3, 1), datetime.date(2015, 3, 16))
(2015, 3, 1)
15 days, 0:00:00
<type 'datetime.timedelta'>
```

DATETIME

```
from datetime import date, datetime, timedelta, time
a = date(2008, 5, 1)
b = time(12, 30, 15)
print(datetime.combine(a, b))
print(datetime(2008, 5, 1, 12, 30, 16))
# parsing and formatting
dt = datetime.strptime("21/11/06 16:30", "%d/%m/%y %H:%M")
print(dt)
print(dt.strftime("%A, %d. %B %Y %I:%M%p"))
```

```
2008-05-01 12:30:15
2008-05-01 12:30:16
2006-11-21 16:30:00
Tuesday, 21. November 2006 04:30PM
```

Formatting codes

There is also a calendar module for e.g. leap year checking, getting days in a month.

MATH MODULE

Mathematical functions and constants.

```
import math
print(math.pi, math.e)
print(math.radians(180), math.degrees(2 * math.pi))
print(math.sin(math.pi / 4))
```

```
(3.141592653589793, 2.718281828459045)
(3.141592653589793, 360.0)
0.707106781187
```

Docs.

These is also cmath for complex numbers.

OS.PATH MODULE

- path manipulation
- Takes care of correct slashes Unix / vs. Windows \

```
import os
p = os.path.join("test", "path", "to", "file.txt") # relative path
print(p)
print(os.path.abspath(p)) # absolute path
# split into path and filename with extension
path, fname_ext = os.path.split(p)
# split filename and extension
fname, extension = os.path.splitext(fname_ext)
print(path, fname_ext, fname, extension)
```

```
test/path/to/file.txt
/media/sf_C/Users/cpaulik/My Dropbox/Arbeit/Teaching/Python Course/for_stud
ents/03-Python Packages/test/path/to/file.txt
('test/path/to', 'file.txt', 'file', '.txt')
```

NAMING IMPORTED PACKAGES

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from scipy import optimize
```

- Check package documentation for recommendations
  - e.g. Scipy guidelines
- mainly up to you
- be consistent

WHERE DOES PYTHON LOOK FOR PACKAGES?

Environment variable PYTHONPATH

- Can be set by
  - Manually in command prompt
  - IDE(e.g. PyScripter)
  - or python script (`sys.path` variable)

Example - Open `WinPython Command Prompt.exe` or your `Shell`

DEMO

use the provided my_code folder as a basis for this demo.

- navigate to the my_code/my_package folder in you shell
- start a python interpreter

```
import hello # this works because current path is searched
hello.function()
hello.variable
exit()
```

THE INIT.PY FILE

- navigate to the my_code folder
- start a python interpreter

```
import my_package.hello as hello # will give error
exit()
```

- make a init.py file in the my_package folder and try again.

CODE IN INIT.PY IS EXECUTED

- fill the `__init__.py` with the following

```
print("code in __init__.py is executed on import")
```

Then try importing again.

```
# this should print the text in __init__.py
import my_package.hello as hello
```

WORKING WITH THE PYTHONPATH

in the my_code directory run

```
python call.py # will throw and ImportError
cd my_package
set PYTHONPATH=%__CD__% # Windows
export PYTHONPATH=$PWD # Unix
cd ..
python call.py # works
```

INSPECTING SYS.PATH

Dynamic changes of `sys.path` are frowned upon

```
import sys
import os
sys.path
sys.path.append(os.path.abspath(".")) # append current path to searchpath
```

THE NAME ATTRIBUTE

- gives the name of the module
- if a .py file is called directly the `__name__` is `__main__`
- see what happens when hello.py is called directly

HOW TO INSTALL A PYTHON PACKAGE

Pure Python Packages

Source code is only python. Easy to install on every platform.

Packages with compiled extensions

These packages often include C or Fortran code that must be compiled for your platform.

- Compiler is needed (not easy on Windows)
- Try to get a precompiled (.whl or .exe) package for your platform. This website hosts precompiled Windows packages.

PACKAGE FORMATS

- zipped (.zip, .tar.gz)
- compiled (.exe, .whl)
- Wheel (.whl) is the new standard packaging format
  - zip file containing all the files needed by a package
  - Also for compiled extensions
  - Example of wheel file format
    - `numpy-1.9.2+mkl-cp34-none-win32.whl`

PIP

- the standard tool for installing packages.
- automatically searches PyPI
- downloads and installs dependencies.

```
pip list # list installed packages
pip install packagename # install
pip install packagename==1.4 # specific version
pip install package1 package2 # multiply packages
pip install packagename -U # upgrade the package to newest version
pip uninstall packagename # uninstall
```

VIRTUAL ENVIRONMENTS

- Isolates packages into standalone environment.
- If applications need different, incompatible versions of libraries.
- If you do not have rights to install packages on system (e.g. VSC)
- Python 2.7 -> `virtualenv`
- Python 3.3, 3.4 -> `pyvenv` included in python

LET'S TRY IT

```
pip install virtualenv # install virtualenv package
virtualenv test_ve
source test_ve/bin/activate
pip install antigravity
pip list
pip uninstall antigravity
pip list
```

```
# Dowload package from PyPI as tar.gz
pip install **.tar.gz
tar -xf
cd
python setup.py install
pip list
```

CREATING YOUR OWN PACKAGE

If you want to distribute your own package it is not too difficult.

- Watch this video
- Use Pyscaffold for project setup
- Use Github for code hosting

ADDITIONAL INFORMATION

- Python Packaging User Guide
- This Talk from PyCon 2014
- Conda package manager good for scientific packages
- Python Distributions come with a lot of packages pre-installed
  - WinPython
  - Anaconda includes conda package manager