http://tw.pycon.org/
2013 Taiwan
PYCON

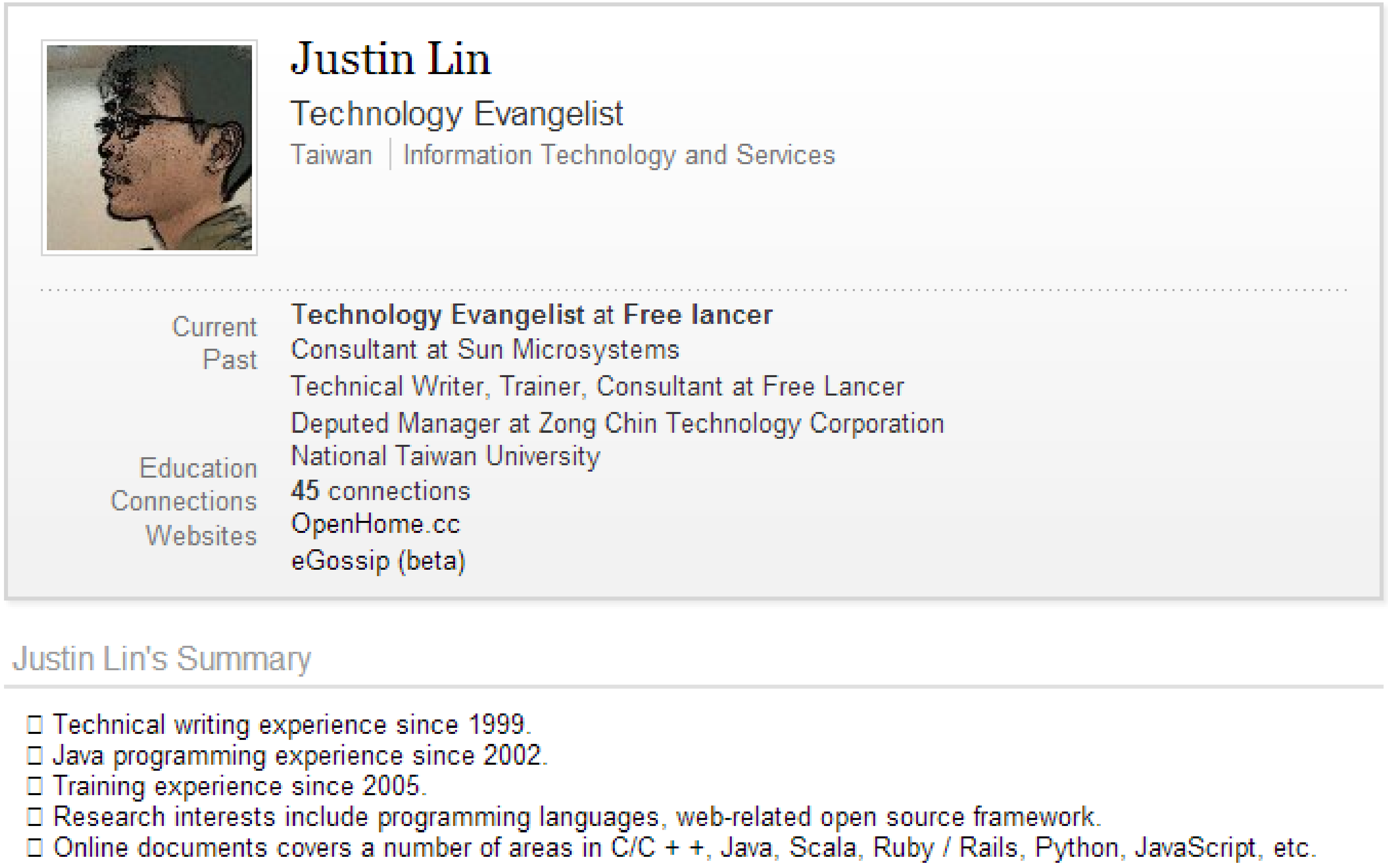
PyCon Taiwan 2013 Tutorial

林信良（Justin Lin）
caterpillar@openhome.cc
http://openhome.cc

Course Objectives

- Learning Python ecosystem
  - languages, tools, libraries…
- Understanding core culture of Python communities
  - coding styles, paradigms, documents, communities …
- Making a connection with PyConTW 2013

Instructor ?



Justin Lin

Technology Evangelist

Taiwan | Information Technology and Services

Current	**Technology Evangelist** at **Free lancer**
Past	Consultant at Sun Microsystems
	Technical Writer, Trainer, Consultant at Free Lancer
	Deputed Manager at Zong Chin Technology Corporation
Education	National Taiwan University
Connections	**45** connections
Websites	OpenHome.cc
	eGossip (beta)

Justin Lin's Summary

- Technical writing experience since 1999.
- Java programming experience since 2002.
- Training experience since 2005.
- Research interests include programming languages, web-related open source framework.
- Online documents covers a number of areas in C/C + +, Java, Scala, Ruby / Rails, Python, JavaScript, etc.

LinkedIn http://www.linkedin.com/in/caterpillar

Student？

- PyCon Taiwan 2013 Tutorial Invitation 收件匣 x

"　．．．

對我而言，要瞭解語言後的文化與生態系，約莫是三到六個月的時間，若以我至三月中前對 Python 生態系的瞭解過程與心得，配合 PyConTW 的議程，將之濃縮為六個小時的課程，你覺得如何？

．．．　"

(Understanding cultures and ecosystem of a language takes me about three to six months. How about wrapping up what I have learned from Python ecosystem before mid-March and considering the agenda of PyConTW to build up a six-hour course?)

Schedule

- The 1st class
  - Preface (currently here)
  - Picking and Installing an Interpreter
    - Implementations
    - Preparing Course Environment
    - Where’re My Libraries?
    - What’s the Relationship among Distutils, Distribute and Pip?
  - Hello! World!
    - Introduction to Unicode Support
    - Basic Input and Output
  - Integrated Development Environment
  - Reference

- The 3rd class
  - The Community
  - Documentation
    - DocStrings
    - Official Documentation
    - PyDoc
    - EpyDoc
  - Data Management Functions
    - Built-in Functions
    - `reduce`
  - Persistence
    - `marshal, pickle, cPickle`
    - DBM
    - `shelve`
    - DB-API 2.0 (PEP 249)
  - References

- The 4th class
  - Libraries vs Frameworks
    - Inversion of Control
    - Do We Need a Framework?
  - Getting Started with Django
    - Creating a Project
    - Creating a Database and an App
    - Playing API with the Python shell
  - Writing Your First View
    - Controller? or Views?
    - URLconf
  - References

- The 5th class
  - Using the Template System
    - Writing Templates
    - Removing Hardcoded URLs in Templates
    - Namespacing URL Names
    - Writing a Simple Form
  - A Bit About CSRF
    - A Cross-Site Request Forgery Example
    - CSRF Countermeasures
  - Testing
    - `assert`
    - `doctest`
  - References

- The 6th class
  - `unittest` (Testing Continued)
    - Test Case
    - Test Fixture
    - Test Suite
    - Test Runner
  - Profiling
    - `timeit`
    - `cProfile` (`profile`)
  - PyCon Taiwan
    - PyCon Taiwan 2012
    - PyCon Taiwan 2013
  - References

Picking and Installing an Interpreter

- 2.x vs 3.x
  - Python 3.0 (a.k.a. "Python 3000" or "Py3k") final was released on **December 3rd, 2008**.
  - Python 3.3.0 was released on September 29th, 2012.
  - Python 2.7.3 was released on **April 9, 2012**.
  - **Python 2.7.x is highly recommended** unless you have a strong reason not to.
  - As more and more modules get ported over to Python3, the easier it will be for others to use it.

Implementations

- **CPython** (http://www.python.org)
  - Is written in C.
  - Compiles Python code to intermediate bytecode.
  - Provides **the highest level of compatibility** with Python packages and C extension modules.
- PyPy (http://pypy.org)
  - Features a **JIT** (just-in-time) compiler.
  - Aims for maximum compatibility with the reference CPython implementation while **improving performance**.

- Jython (http://www.jython.org/)
  - An implementation of Python for the **JVM**.
  - Compiles Python code to **Java byte code.**
  - Can import and use any Java class the same as a Python module.
- IronPython (http://ironpython.net/)
  - An open-source implementation of the Python programming language which is tightly integrated with the **.NET Framework**.
  - Can use the .NET Framework and Python libraries.
  - Other .NET languages can use Python code just as easily.

Preparing Course Environment

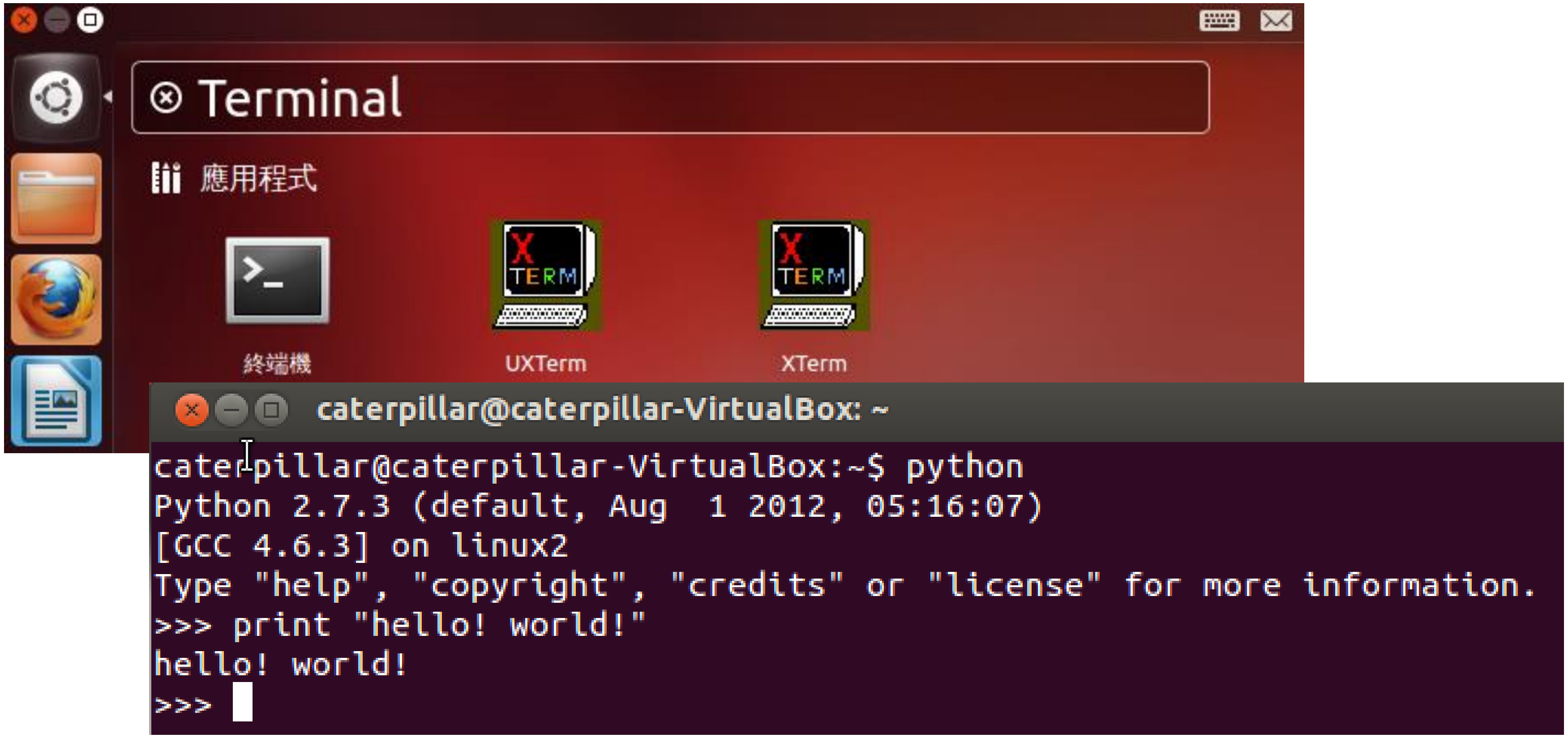
- Ubuntu 12.04 LTS
- The Slide and lab file.

```
sudo apt-get install git
git clone https://github.com/JustinSDK/PyConTW2013Tutorial.git
```

- **Python 2.7.3**
  - Distribute
  - Pip
  - Virtualenv

Exercise 0

- **Installing Python 2.7.3**
- Ubuntu 12.04 comes with Python 2.7.3 out of the box.
- All you have to do is to open a terminal and `python`!



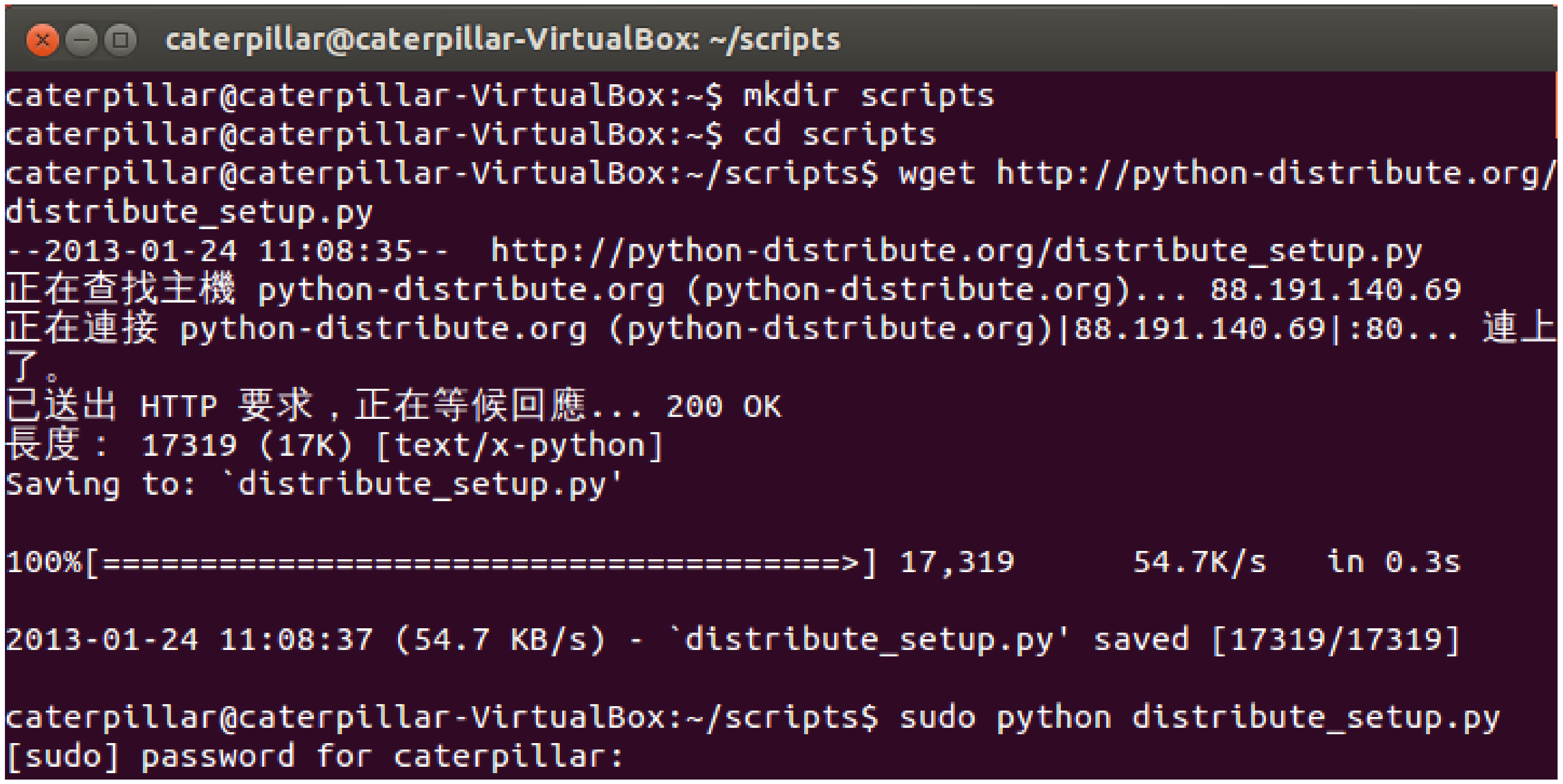
```
caterpillar@caterpillar-VirtualBox:~$ python
Python 2.7.3 (default, Aug  1 2012, 05:16:07)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> print "hello! world!"
hello! world!
>>> 
```

Exercise 1

- **Installing Distribute, Pip and Virtualenv**
- **Distribute** extends the packaging and installation facilities provided by the **distutils** in the standard library.
  - run the python script available below:
    http://python-distribute.org/distribute_setup.py

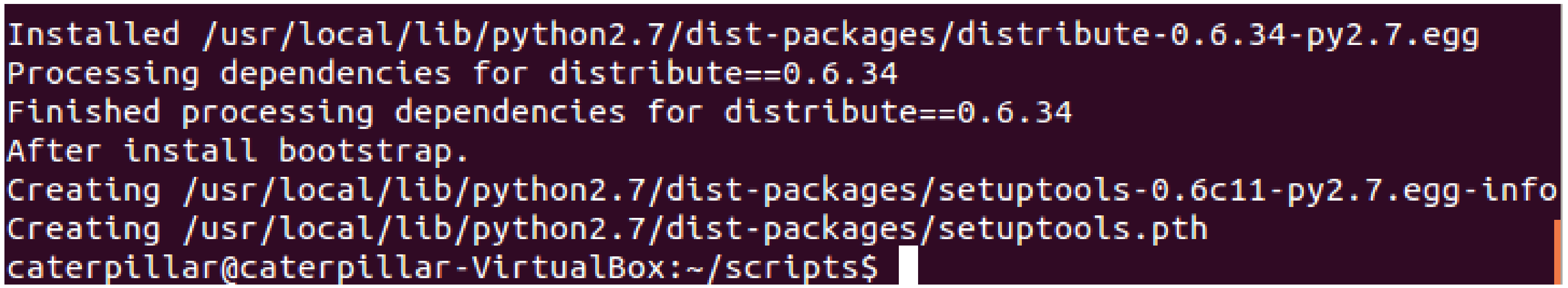
```
~$ mkdir scripts
~$ cd scripts
~/scripts$ wget http://python-distribute.org/distribute_setup.py
~/scripts$ sudo python distribute_setup.py
```

What You Should See



```
caterpillar@caterpillar-VirtualBox:~$ mkdir scripts
caterpillar@caterpillar-VirtualBox:~$ cd scripts
caterpillar@caterpillar-VirtualBox:~/scripts$ wget http://python-distribute.org/
distribute_setup.py
--2013-01-24 11:08:35--  http://python-distribute.org/distribute_setup.py
正在查找主機 python-distribute.org (python-distribute.org)... 88.191.140.69
正在連接 python-distribute.org (python-distribute.org)|88.191.140.69|:80... 連上
了。
已送出 HTTP 要求，正在等候回應... 200 OK
長度： 17319 (17K) [text/x-python]
Saving to: `distribute_setup.py'

100%[======================================>] 17,319      54.7K/s   in 0.3s

2013-01-24 11:08:37 (54.7 KB/s) - `distribute_setup.py' saved [17319/17319]

caterpillar@caterpillar-VirtualBox:~/scripts$ sudo python distribute_setup.py
[sudo] password for caterpillar:
```

~

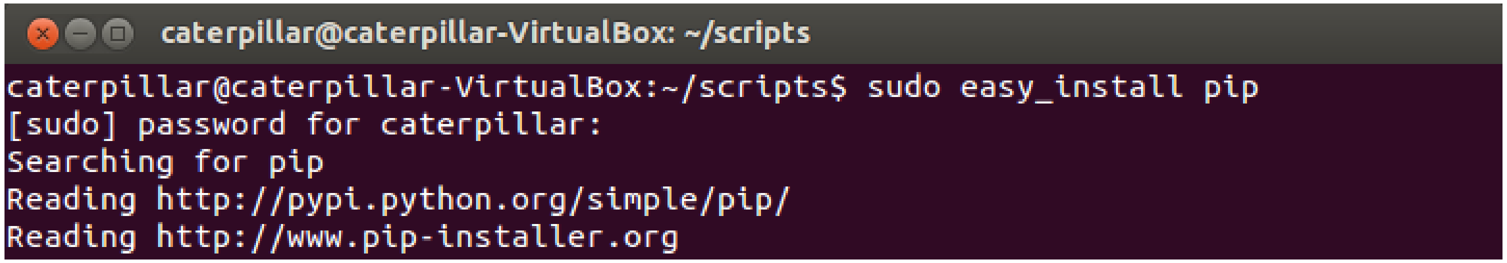


```
Installed /usr/local/lib/python2.7/dist-packages/distribute-0.6.34-py2.7.egg
Processing dependencies for distribute==0.6.34
Finished processing dependencies for distribute==0.6.34
After install bootstrap.
Creating /usr/local/lib/python2.7/dist-packages/setuptools-0.6c11-py2.7.egg-info
Creating /usr/local/lib/python2.7/dist-packages/setuptools.pth
caterpillar@caterpillar-VirtualBox:~/scripts$
```

- The new``easy_install`` command you have available is considered by many to be deprecated, so we will install its replacement: **pip**.
- The **virtualenv** kit provides the ability to create virtual Python environments that do not interfere with either each other, or the main Python installation.

```
~/scripts$ sudo easy_install pip
~/scripts$ sudo pip install virtualenv
```

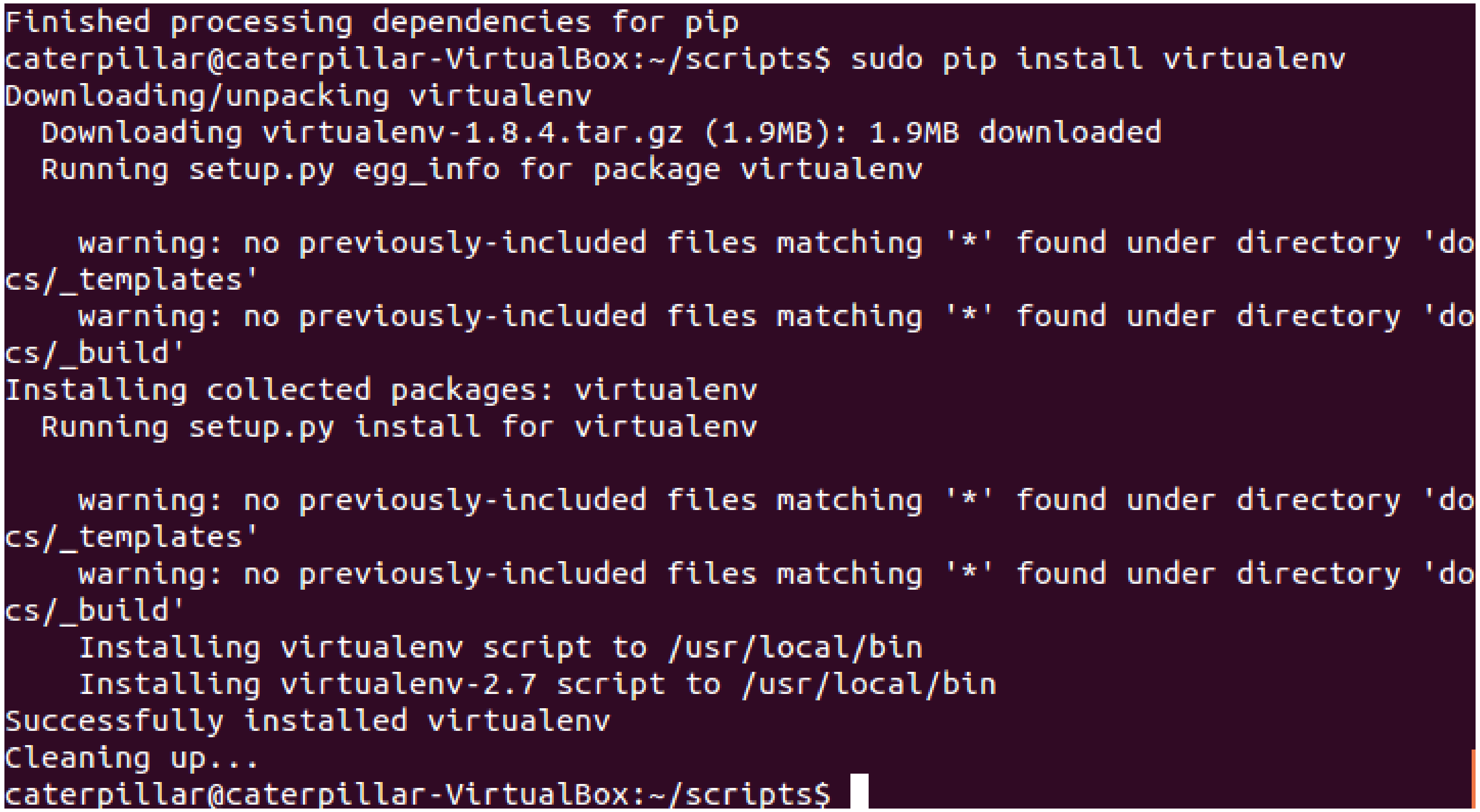
What You Should See



caterpillar@caterpillar-VirtualBox: ~/scripts

```
caterpillar@caterpillar-VirtualBox:~/scripts$ sudo easy_install pip
[sudo] password for caterpillar:
Searching for pip
Reading http://pypi.python.org/simple/pip/
Reading http://www.pip-installer.org
```

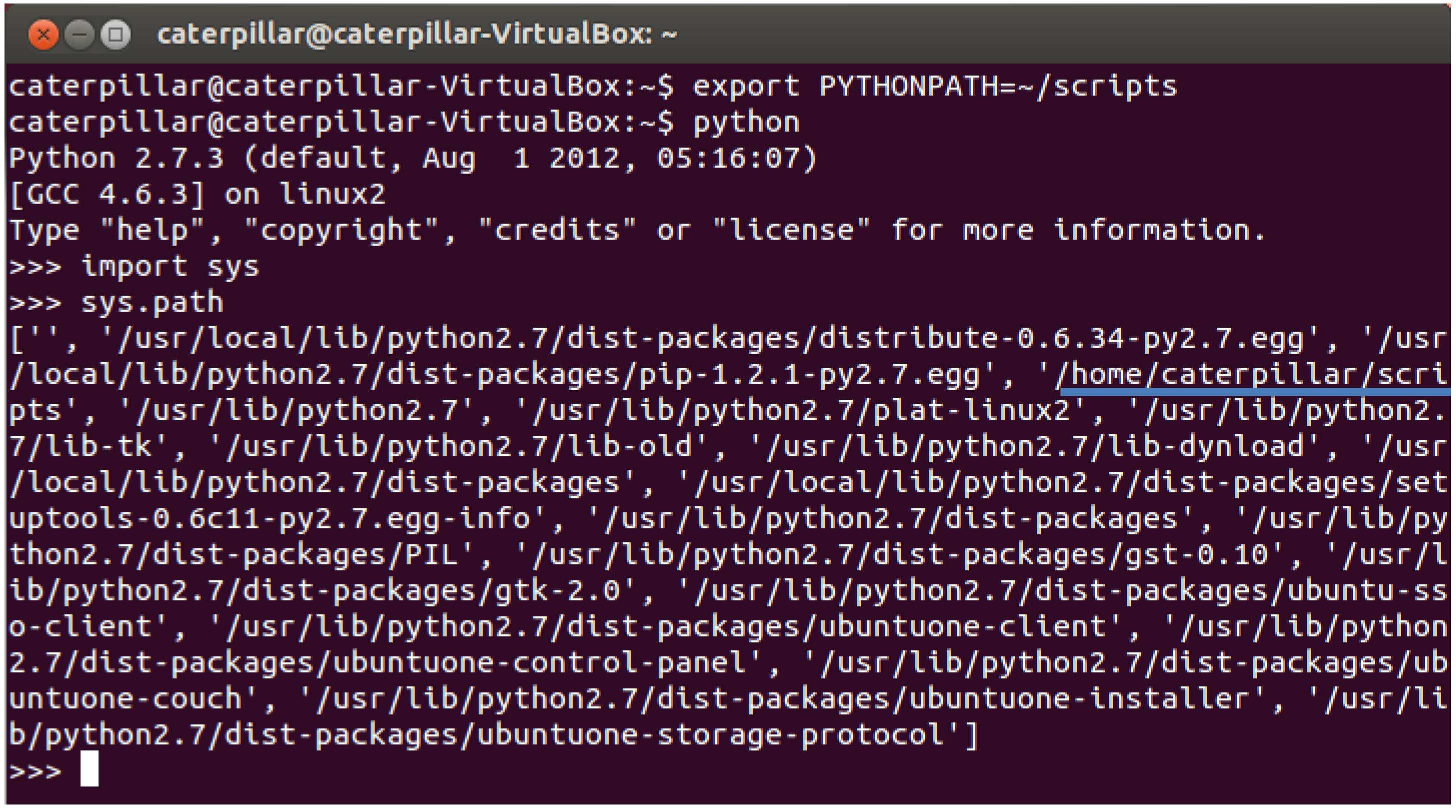
~



```
Finished processing dependencies for pip
caterpillar@caterpillar-VirtualBox:~/scripts$ sudo pip install virtualenv
Downloading/unpacking virtualenv
  Downloading virtualenv-1.8.4.tar.gz (1.9MB): 1.9MB downloaded
  Running setup.py egg_info for package virtualenv

    warning: no previously-included files matching '*' found under directory 'do
cs/_templates'
    warning: no previously-included files matching '*' found under directory 'do
cs/_build'
Installing collected packages: virtualenv
  Running setup.py install for virtualenv

    warning: no previously-included files matching '*' found under directory 'do
cs/_templates'
    warning: no previously-included files matching '*' found under directory 'do
cs/_build'
    Installing virtualenv script to /usr/local/bin
    Installing virtualenv-2.7 script to /usr/local/bin
Successfully installed virtualenv
Cleaning up...
caterpillar@caterpillar-VirtualBox:~/scripts$
```

Where're My Libraries?

- The **`sys.path`** is a list of strings that specifies the search path for modules.
- Use the environment variable **`PYTHONPATH`** to augment the default search path for module files.



caterpillar@caterpillar-VirtualBox: ~

```
caterpillar@caterpillar-VirtualBox:~$ export PYTHONPATH=~/scripts
caterpillar@caterpillar-VirtualBox:~$ python
Python 2.7.3 (default, Aug  1 2012, 05:16:07)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import sys
>>> sys.path
['', '/usr/local/lib/python2.7/dist-packages/distribute-0.6.34-py2.7.egg', '/usr
/local/lib/python2.7/dist-packages/pip-1.2.1-py2.7.egg', '/home/caterpillar/scri
pts', '/usr/lib/python2.7', '/usr/lib/python2.7/plat-linux2', '/usr/lib/python2.
7/lib-tk', '/usr/lib/python2.7/lib-old', '/usr/lib/python2.7/lib-dynload', '/usr
/local/lib/python2.7/dist-packages', '/usr/local/lib/python2.7/dist-packages/set
uptools-0.6c11-py2.7.egg-info', '/usr/lib/python2.7/dist-packages', '/usr/lib/py
thon2.7/dist-packages/PIL', '/usr/lib/python2.7/dist-packages/gst-0.10', '/usr/l
ib/python2.7/dist-packages/gtk-2.0', '/usr/lib/python2.7/dist-packages/ubuntu-ss
o-client', '/usr/lib/python2.7/dist-packages/ubuntuone-client', '/usr/lib/python
2.7/dist-packages/ubuntuone-control-panel', '/usr/lib/python2.7/dist-packages/ub
untuone-couch', '/usr/lib/python2.7/dist-packages/ubuntuone-installer', '/usr/li
b/python2.7/dist-packages/ubuntuone-storage-protocol']
>>> 
```

What’s the Relationship among Distutils, Setuptools, Distribute and Pip?

- Distutils
  - The Python standard library for building and installing additional modules.
  - For simple installation scenarios.
  - Basic steps:
    - Untar the downloaded file (e.g. tar xzvf Django-X.Y.tar.gz)
    - Change into the directory. Basically, all you need is **setup.py**.
    - `sudo python setup.py install`
- Setuptools
  - Extends distutils.
  - Is de facto standard of Python community.
  - Has problems of slow development, messy code…

- Distribute
  - Extends distutils.
  - Is intended to **replace Setuptools** as the standard method for working with Python module distributions.
  - Provides **a backward compatible** version to replace Setuptools and makes all distributions that depend on Setuptools work as before.
  - So, once setuptools or distribute is installed, `easy_install` is prepared.
  - The `easy_install` command is considered by many to be deprecated due to lack of unstallation command, svn-only support…
- Pip
  - **An `easy_install` replacement.**
  - Allows for uninstallation of packages, and is actively maintained, unlike `easy_install`.
  - Virtualenv is its good partner.
  - Basic commands:
    - `pip install [PACKAGE_NAME]`
    - `pip unstall [PACKAGE_NAME]`

Hello! World!

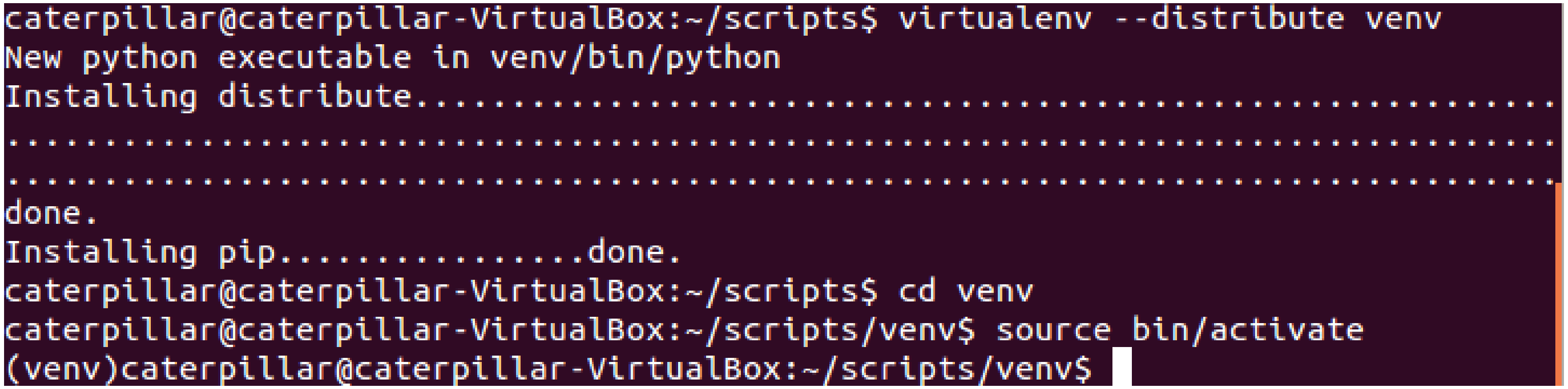
- The **virtualenv** kit provides the ability to create virtual Python environments that do not interfere with either each other, or the main Python installation.
- Create a virtual Python environment:
  - `virtualenv --distribute venv`
- Activate the environment:
  - `source bin/activate`
- Deactivate the environment:
  - `deactivate`

Exercise 2

- **Create and activate a virtual Python environment.**
- **Prompt a use to provide a filename, read the file and print the content in the terminal. Consider the character encoding problems.**

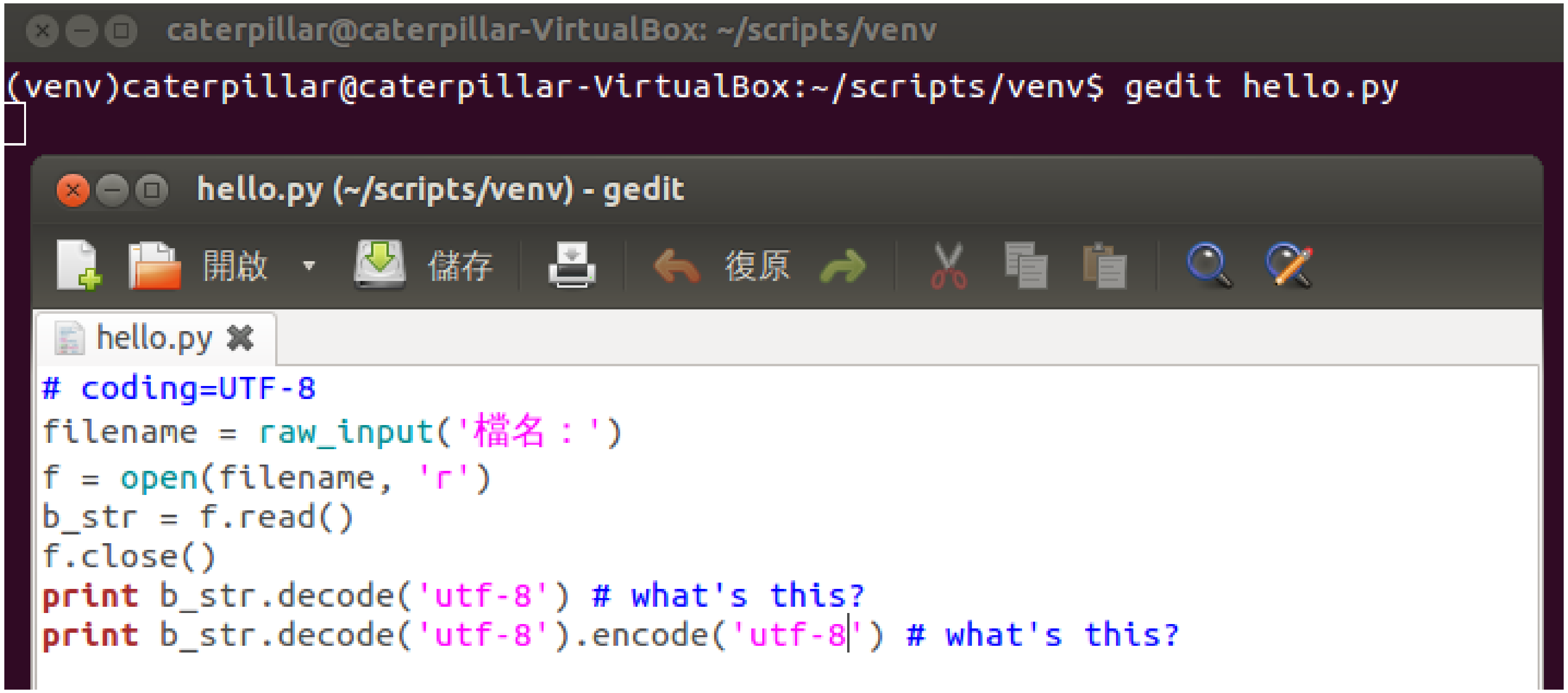
```
~/scripts$ virtualenv --distribute venv
~/scripts$ cd venv
~/scripts/venv$ source bin/activate
```

What You Should See



```
caterpillar@caterpillar-VirtualBox:~/scripts$ virtualenv --distribute venv
New python executable in venv/bin/python
Installing distribute.................................................................
..................................................................................
..................................................................................
done.
Installing pip................done.
caterpillar@caterpillar-VirtualBox:~/scripts$ cd venv
caterpillar@caterpillar-VirtualBox:~/scripts/venv$ source bin/activate
(venv)caterpillar@caterpillar-VirtualBox:~/scripts/venv$ 
```

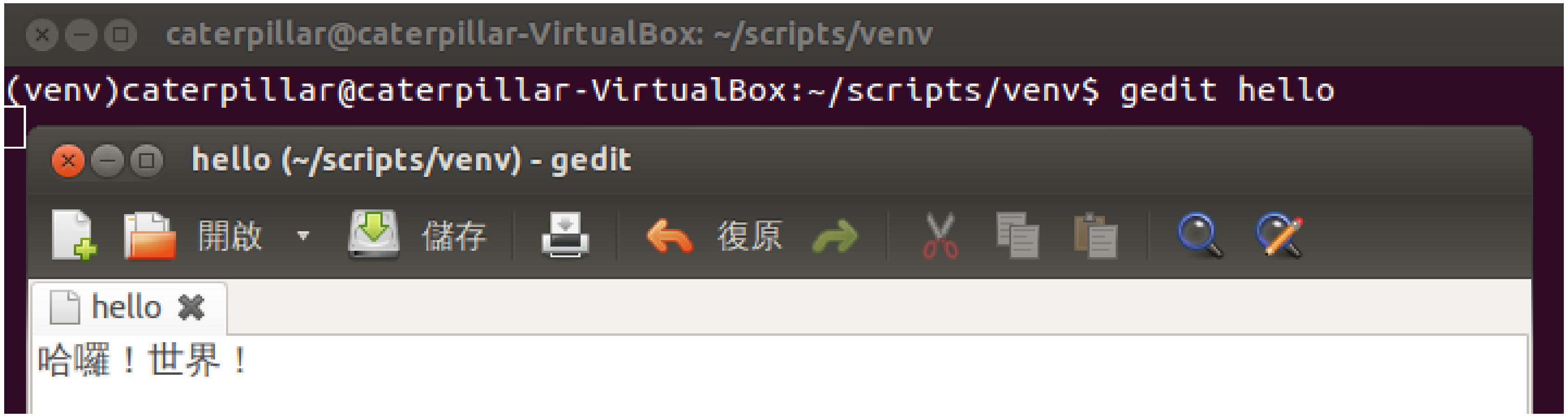
~/scripts/venv$ **gedit hello.py**


caterpillar@caterpillar-VirtualBox: ~/scripts/venv
hello.py (~/scripts/venv) - gedit
開啟
儲存
復原
hello.py

```
(venv)caterpillar@caterpillar-VirtualBox:~/scripts/venv$ gedit hello.py
```

```
# coding=UTF-8
filename = raw_input('檔名：')
f = open(filename, 'r')
b_str = f.read()
f.close()
print b_str.decode('utf-8') # what's this?
print b_str.decode('utf-8').encode('utf-8') # what's this?
```

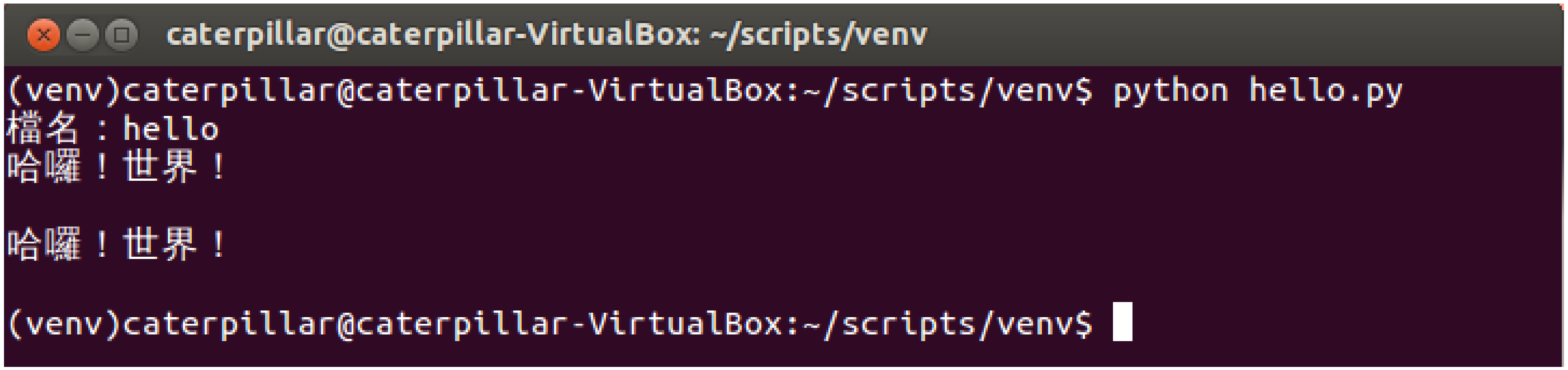
~/scripts/venv$ **gedit hello**


caterpillar@caterpillar-VirtualBox: ~/scripts/venv
hello (~/scripts/venv) - gedit
開啟
儲存
復原
hello

```
(venv)caterpillar@caterpillar-VirtualBox:~/scripts/venv$ gedit hello
```

```
哈囉！世界！
```

What You Should See



```
(venv)caterpillar@caterpillar-VirtualBox:~/scripts/venv$ python hello.py
檔名：hello
哈囉！世界！

哈囉！世界！

(venv)caterpillar@caterpillar-VirtualBox:~/scripts/venv$ 
```

Introduction to Unicode Support

- Default encoding of Ubuntu: UTF-8.
- Python 2:
  - Strings are actual byte sequence representing the data.

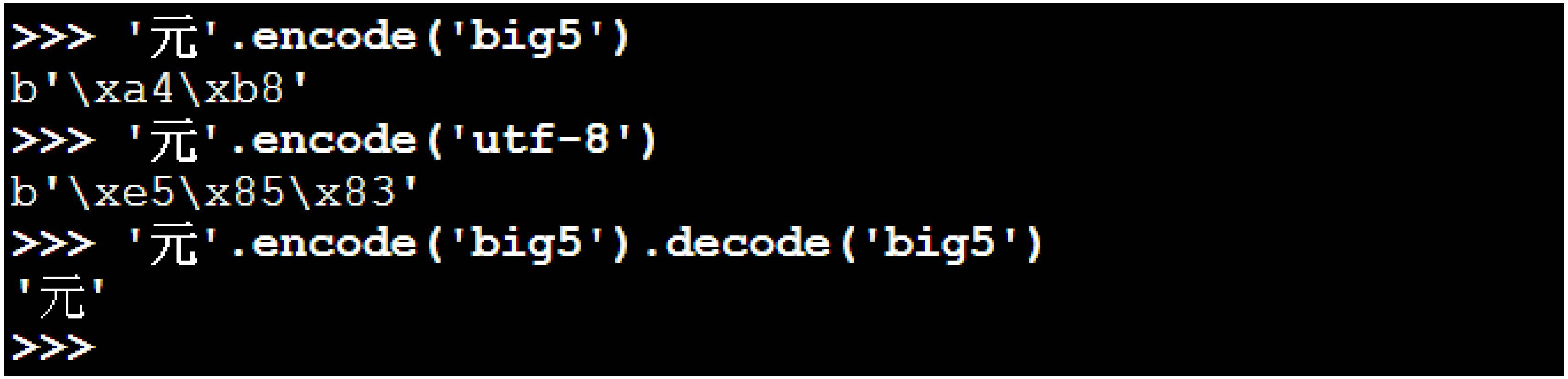
encoding declaration

```
# coding=UTF-8
text = '測試'
print len(text) # print "6"
```

  - Unicode literals are written as strings prefixed with the `'u'` or `'U'` character

```
# coding=UTF-8
text = u'測試'
print type(text) # print "<type 'unicode'>"
print len(text)  # print "2"
```

- Python 2:
  - `decode` interprets the string using the given encoding and returns a `unicode` instance.
  - `encode` returns an 8-bit string version of the Unicode string.
- Python 3: Unicode by default.
  - `decode` returns a `bytes` instance representing byte sequence.
  - `encode` returns a `str` instance representing the Unicode string.



```
>>> '元'.encode('big5')
b'\xa4\xb8'
>>> '元'.encode('utf-8')
b'\xe5\x85\x83'
>>> '元'.encode('big5').decode('big5')
'元'
>>>
```

Basic Input and Output

- Read a file:

```
import sys
file = open(sys.argv[1], 'r')
content = file.read()
print content
file.close()
```

Import a module

Command line arguments

- Write a file:

```
import sys
file = open(sys.argv[1], 'w')
file.write('test')
file.close()
```

- Three ways for reading all content in a file:

```
import sys
file = open(sys.argv[1], 'r')
while True:
    line = file.readline()
    if not line: break
    print line
file.close()
```

```
import sys
file = open(sys.argv[1], 'r')
for line in file.readlines():
    print line
file.close()
```

```
import sys
for line in open(sys.argv[1], 'r'):
    print line
```

Integrated Development Environment

- Sometimes, it‘s just the problem of flavor.
  - PyCharm / IntelliJ IDEA
    - http://www.jetbrains.com/pycharm/
  - PyDev / Eclipse plugin
    - http://pydev.org/
  - Komodo IDE
    - http://www.activestate.com/komodo-ide
  - Spyder
    - http://code.google.com/p/spyderlib/
  - WingIDE
    - http://wingware.com/
  - NINJA-IDE
    - http://www.ninja-ide.org/
  - Python Tools for Visual Studio
    - http://pytools.codeplex.com/

References

- Implementations
  - http://www.python.org/download/releases/3.0/
  - http://www.python.org/download/releases/2.7.3/
  - http://docs.python-guide.org/en/latest/starting/which-python/
- Preparing course environment
  - http://docs.python-guide.org/en/latest/starting/install/linux/
- Where're my libraries?
  - http://docs.python.org/2/using/cmdline.html
- What's the relationship among distutils, Distribute and Pip?
  - http://docs.python.org/2/library/distutils.html
  - http://pypi.python.org/pypi/distribute
  - http://pypi.python.org/pypi/pip
  - http://blog.yangyubo.com/2012/07/27/python-packaging/
  - http://www.openfoundry.org/tw/tech-column/8536-introduction-of-python-extension-management-tools
- Hello! World!
  - http://caterpillar.onlyfun.net/Gossip/Python/IOABC.html
  - http://caterpillar.onlyfun.net/Gossip/Encoding/
  - http://caterpillar.onlyfun.net/Gossip/Encoding/Python.html

Learning Python Language

- What’re the most essential elements of a language?

Algorithms + Data Structures = Programs

-- Niklaus E. Wirth -- The chief designer of Pascal

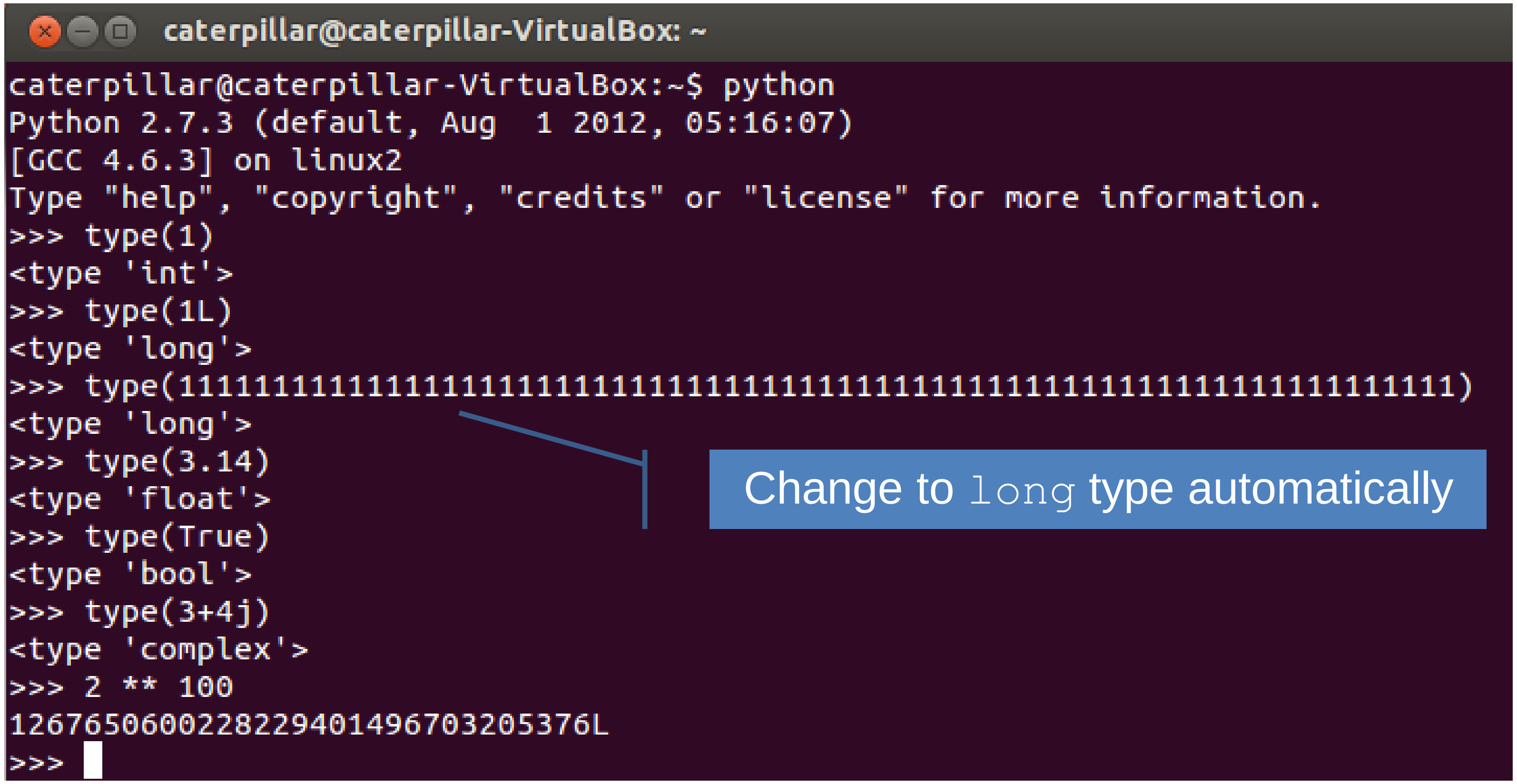
- How to encapsulate your code?
- Focus on the essence of Python, not nuts and bolts.
  - Built-in types, variables and operators
  - Functions, classes and modules

Built-in Types

- **Every thing is an object.**
  - Python, however, does not impose object-oriented programming as the main programming paradigm.
- Numerical types
  - `int, long, float, bool, complex`
- String type
- Container types
  - `list`, `set`, `dict`, `tuple`

Numerical Types

- `int, long, float, bool, complex`
- The `type` function returns the type of any object.



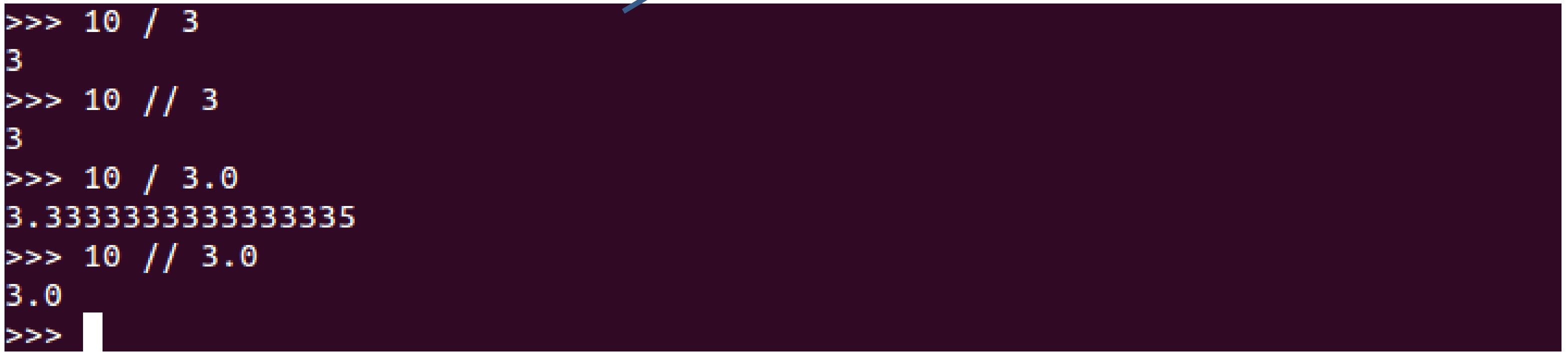
```
caterpillar@caterpillar-VirtualBox:~$ python
Python 2.7.3 (default, Aug  1 2012, 05:16:07)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> type(1)
<type 'int'>
>>> type(1L)
<type 'long'>
>>> type(1111111111111111111111111111111111111111111111111111111111111111111111)
<type 'long'>
>>> type(3.14)
<type 'float'>
>>> type(True)
<type 'bool'>
>>> type(3+4j)
<type 'complex'>
>>> 2 ** 100
1267650600228229401496703205376L
>>> 
```

Change to `long` type automatically

What You Should Know

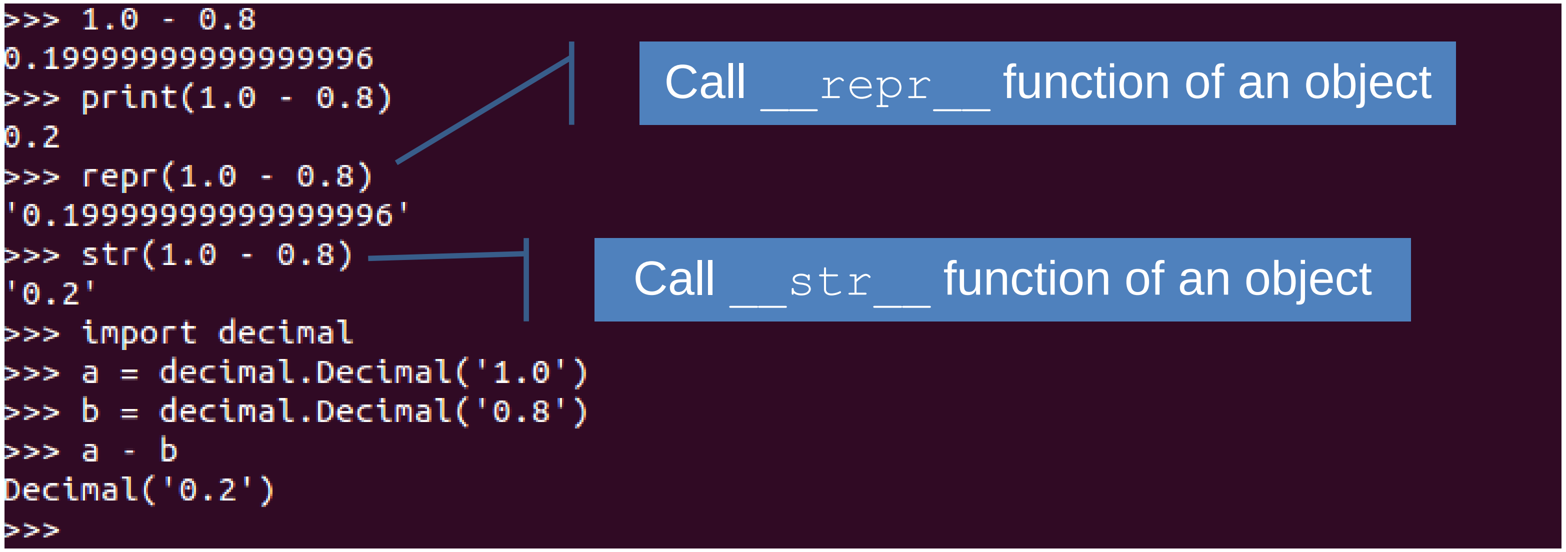
- Python float division:

Different results in different versions



```
>>> 10 / 3
3
>>> 10 // 3
3
>>> 10 / 3.0
3.3333333333333335
>>> 10 // 3.0
3.0
>>>
```

- Float decision, `repr` and `str`:



```
>>> 1.0 - 0.8
0.19999999999999996
>>> print(1.0 - 0.8)
0.2
>>> repr(1.0 - 0.8)
'0.19999999999999996'
>>> str(1.0 - 0.8)
'0.2'
>>> import decimal
>>> a = decimal.Decimal('1.0')
>>> b = decimal.Decimal('0.8')
>>> a - b
Decimal('0.2')
>>>
```

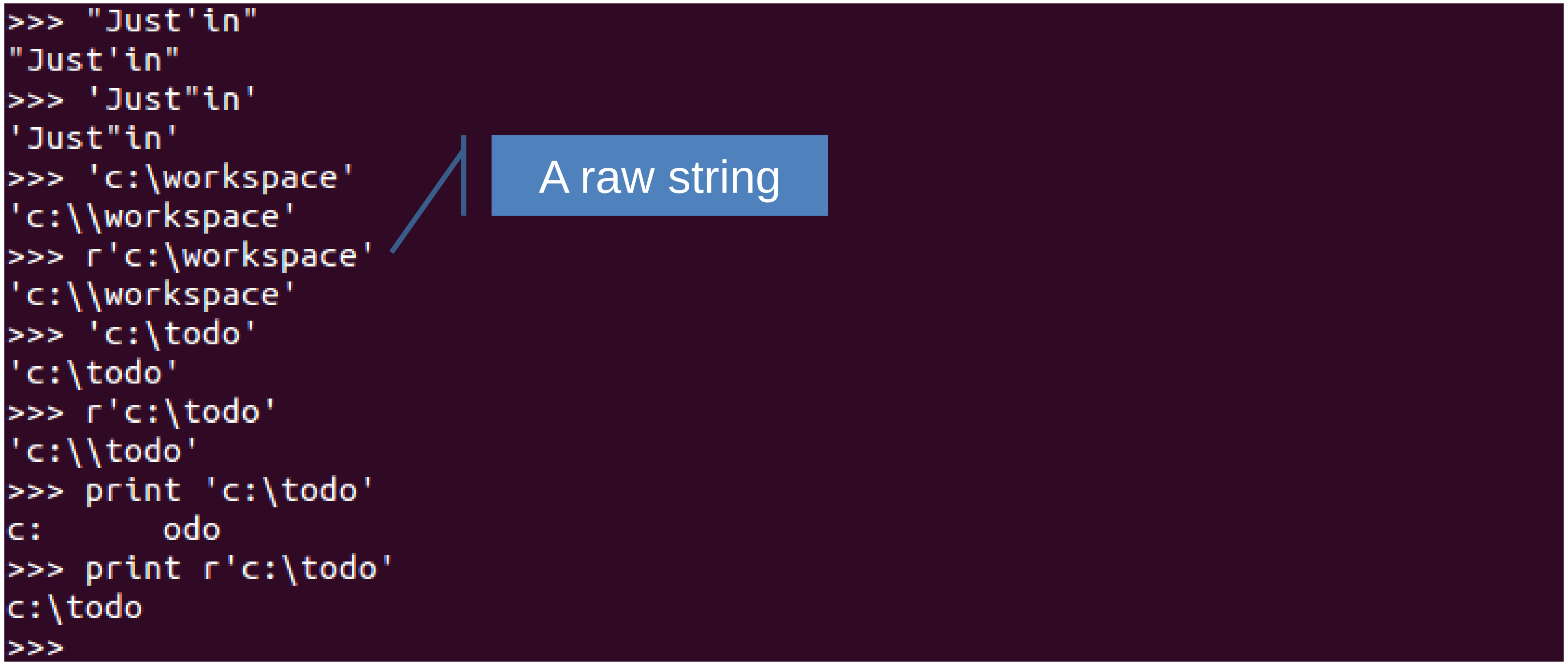
Call `__repr__` function of an object

Call `__str__` function of an object

- `__repr__` computes the “official” string representation of an object.
- `__str__` compute the “informal” string representation of an object.
- **`__repr__` is to be unambigous and `__str__` is to be readable.**
- The `decimal` module provides support for decimal floating point arithmetic.

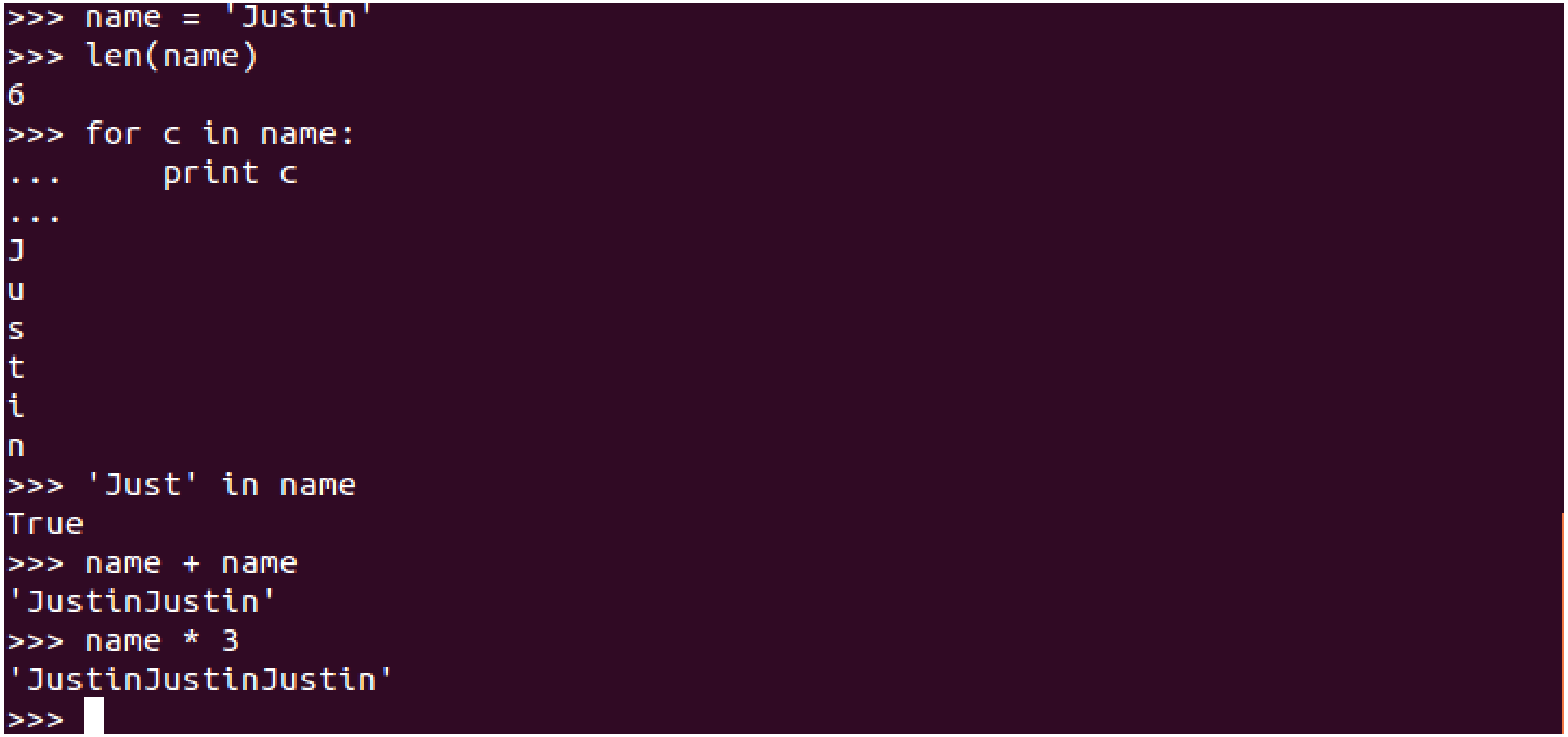
String Type

- `''` and `""` are the same in Python and replaceable.
- Use a raw string if you want to represent `'\'` itself.



```
>>> "Just'in"
"Just'in"
>>> 'Just"in'
'Just"in'
>>> 'c:\workspace'
'c:\\workspace'
>>> r'c:\workspace'
'c:\\workspace'
>>> 'c:\todo'
'c:\todo'
>>> r'c:\todo'
'c:\\todo'
>>> print 'c:\todo'
c:      odo
>>> print r'c:\todo'
c:\todo
>>>
```

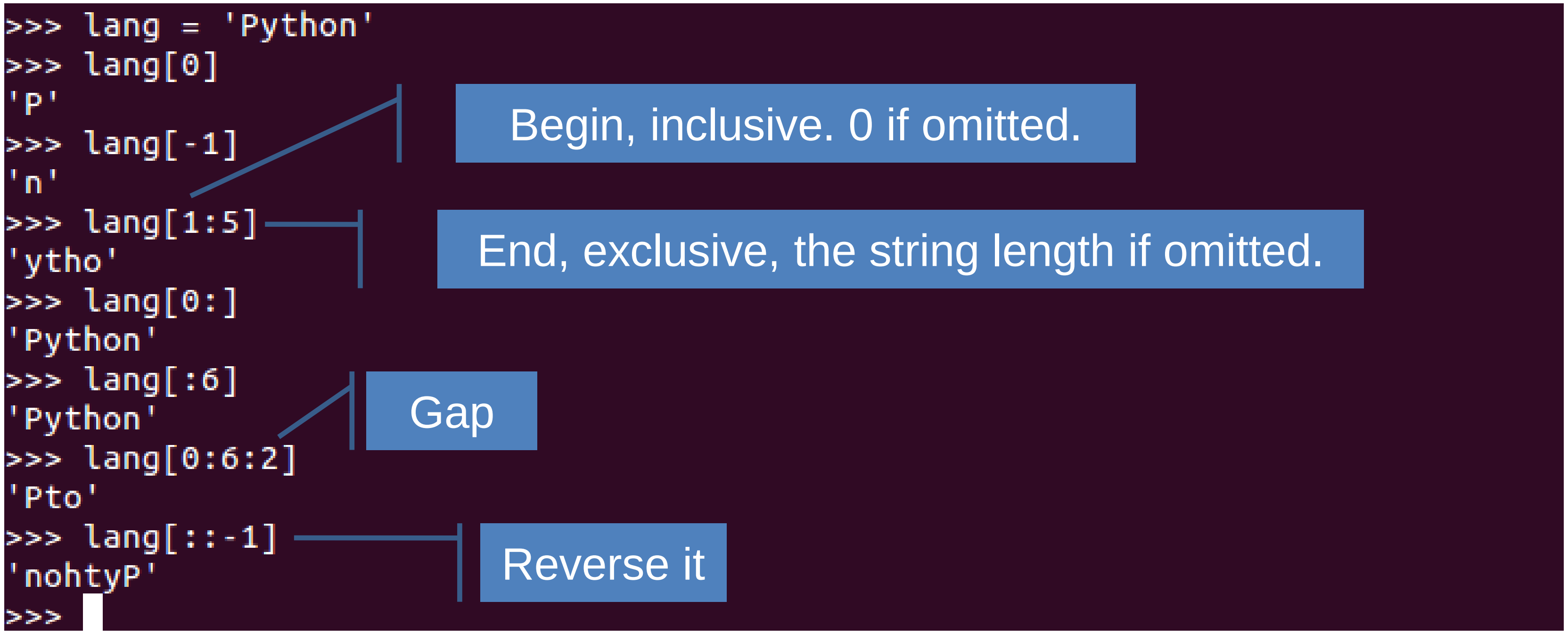
- A string is **immutable**.
- `len` returns the string length. Use `for` to iterate a string. `in` tests if a string contains a substring. `+` is for concatenating two strings. `*` replicates a string.



```
>>> name = 'Justin'
>>> len(name)
6
>>> for c in name:
...     print c
...
J
u
s
t
i
n
>>> 'Just' in name
True
>>> name + name
'JustinJustin'
>>> name * 3
'JustinJustinJustin'
>>> 
```

String Slicing

- `[]` can specified an index to get a character from a string. A negative index is counted from the last element.
- **The most useful power of `[]` is slicing.**

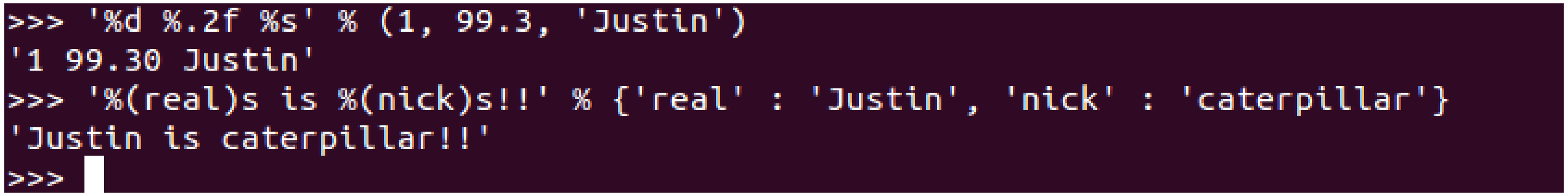


```
>>> lang = 'Python'
>>> lang[0]
'P'
>>> lang[-1]
'n'
>>> lang[1:5]
'ytho'
>>> lang[0:]
'Python'
>>> lang[:6]
'Python'
>>> lang[0:6:2]
'Pto'
>>> lang[::-1]
'nohtyP'
>>>
```

Begin, inclusive. 0 if omitted.
End, exclusive, the string length if omitted.
Gap
Reverse it

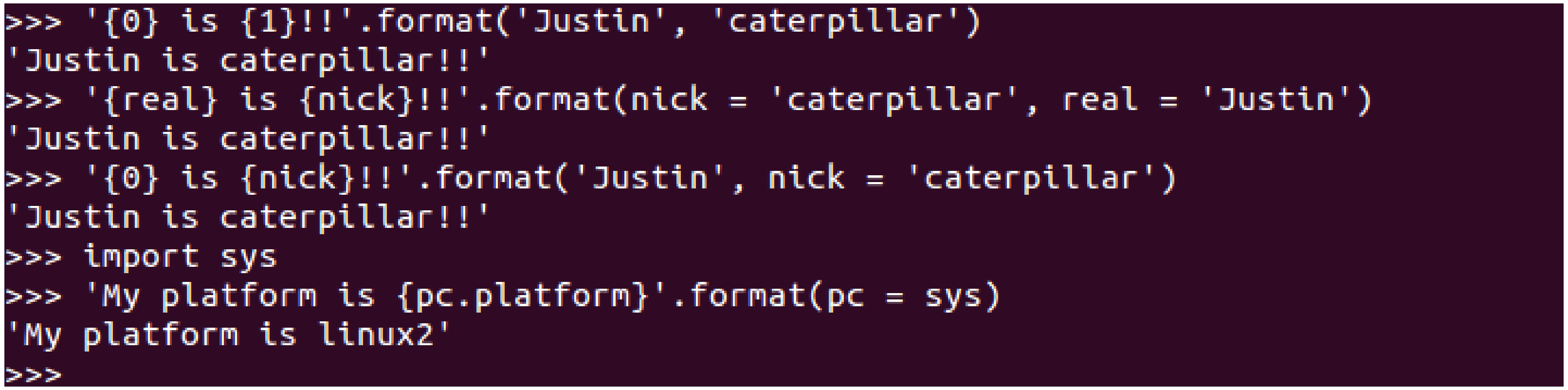
String Formatting

- Old String Formatting Operations



```
>>> '%d %.2f %s' % (1, 99.3, 'Justin')
'1 99.30 Justin'
>>> '%(real)s is %(nick)s!!' % {'real' : 'Justin', 'nick' : 'caterpillar'}
'Justin is caterpillar!!'
>>>
```

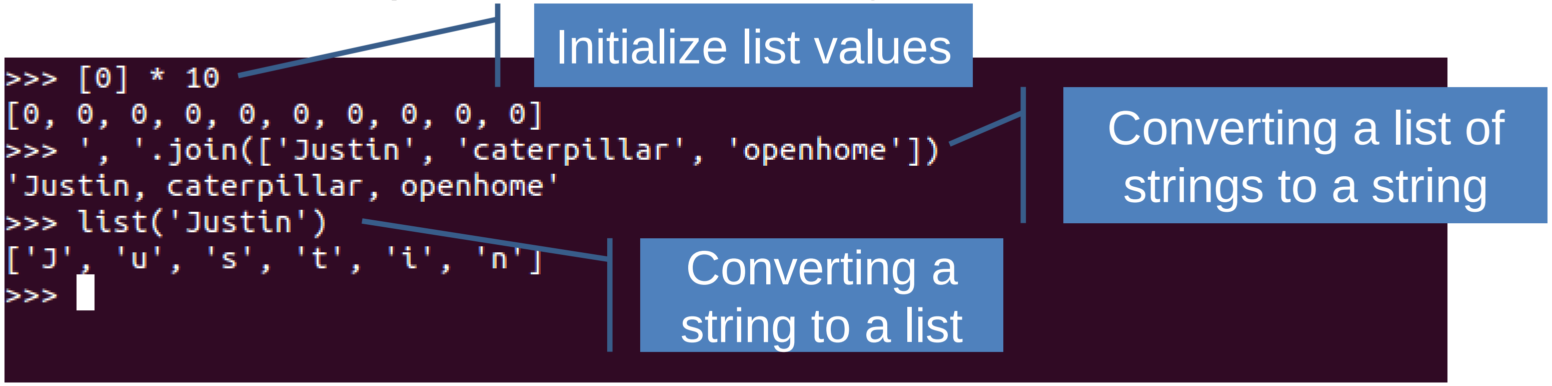
- New String Formatting Operations (after Python 2.6)



```
>>> '{0} is {1}!!'.format('Justin', 'caterpillar')
'Justin is caterpillar!!'
>>> '{real} is {nick}!!'.format(nick = 'caterpillar', real = 'Justin')
'Justin is caterpillar!!'
>>> '{0} is {nick}!!'.format('Justin', nick = 'caterpillar')
'Justin is caterpillar!!'
>>> import sys
>>> 'My platform is {pc.platform}'.format(pc = sys)
'My platform is linux2'
>>>
```

List Type

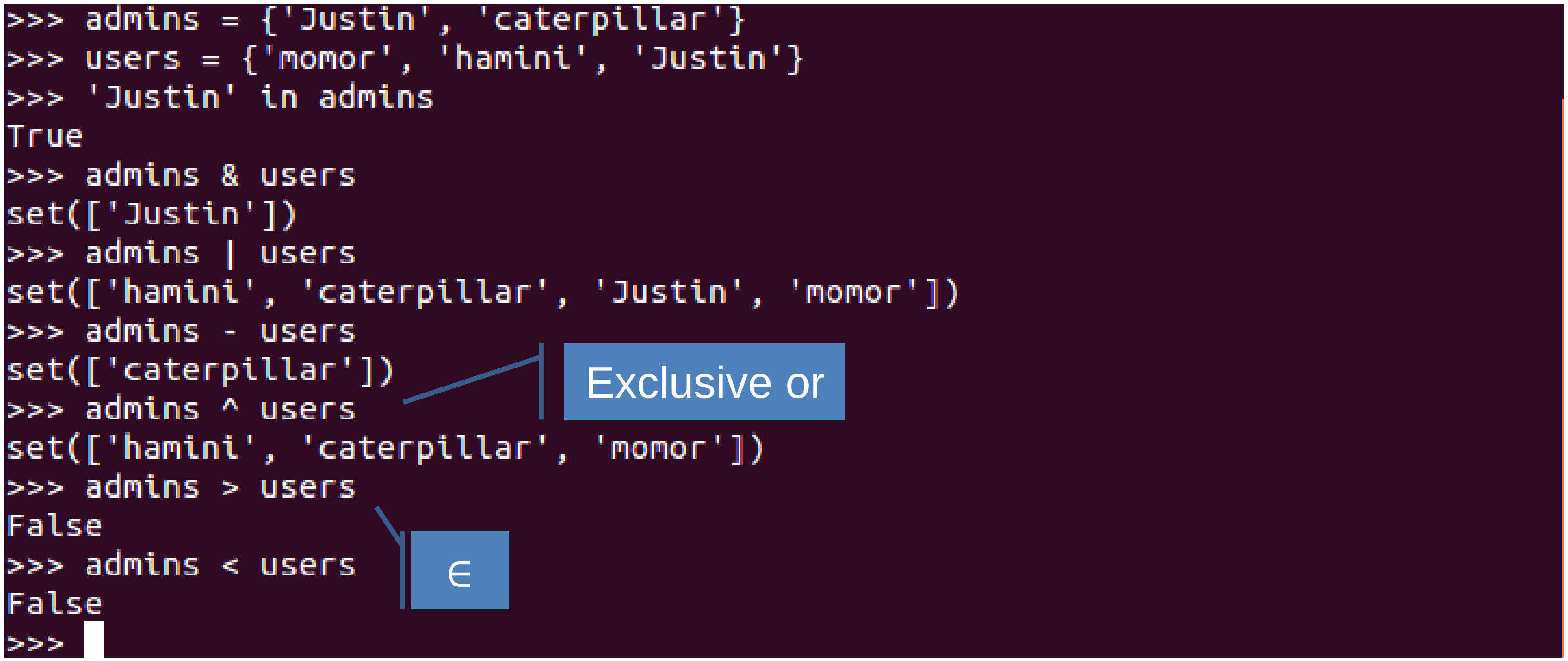
- An ordered and **mutable** collection.
  - `[1, 2, 3]` creates a list with elements `1`, `2`, and `3` in the index `0`, `1` and `2`.
- Shares common operations with strings.
  - `len` returns the list length. Use `for` to iterate a list. `in` tests if a list contains an element. `+` is for concatenating two lists. `*` replicates a list.
  - `[]` can specified an index to get a character from a string. A negative index is counted from the last element.
  - The most useful power of `[]` is slicing.


Initialize list values
Converting a list of strings to a string
Converting a string to a list

```
>>> [0] * 10
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
>>> ', '.join(['Justin', 'caterpillar', 'openhome'])
'Justin, caterpillar, openhome'
>>> list('Justin')
['J', 'u', 's', 't', 'i', 'n']
>>> 
```

Set Type

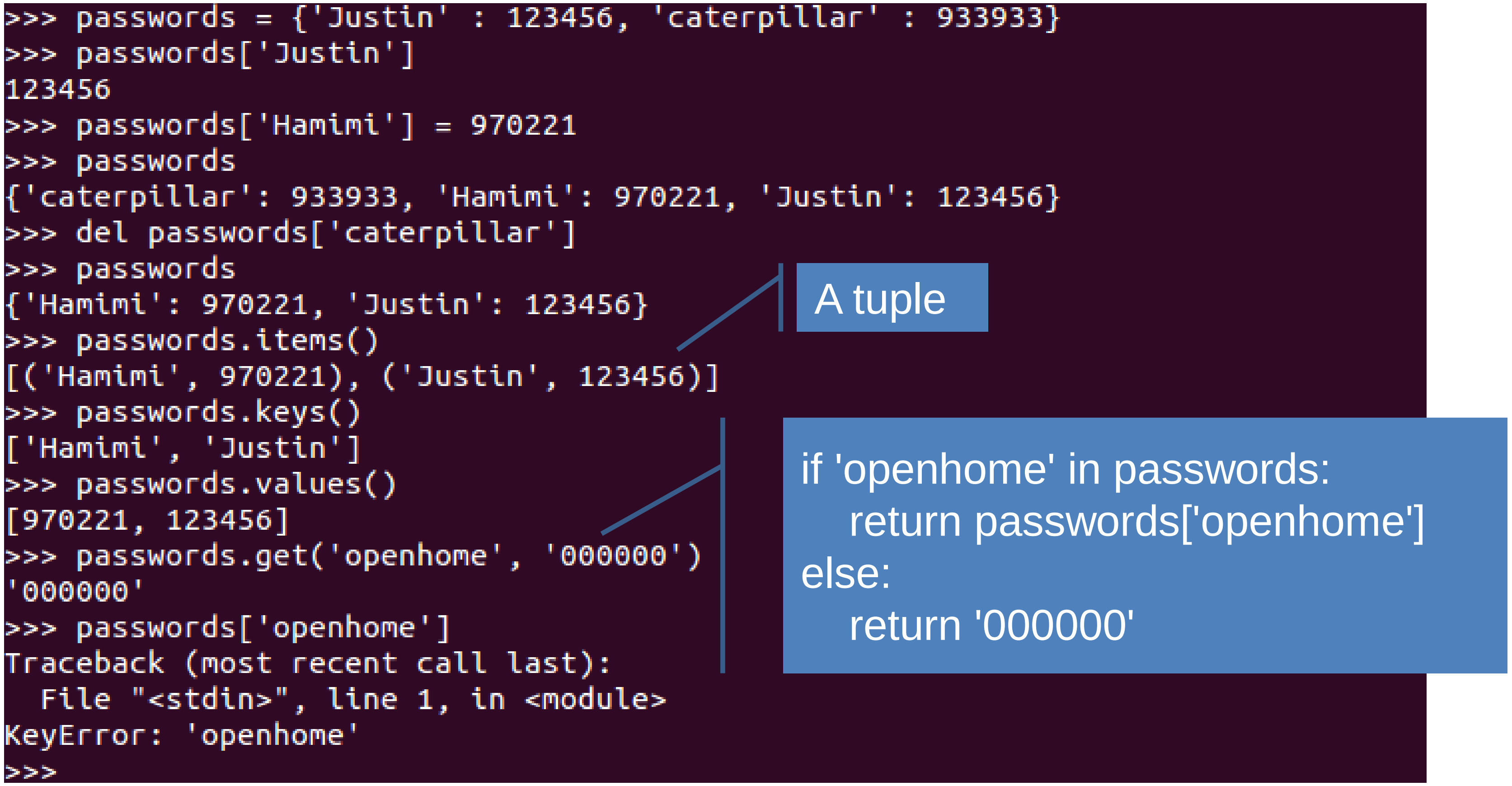
- A unordered collection. Contains no duplicate elements.
- Elements should be **immutable**.



```
>>> admins = {'Justin', 'caterpillar'}
>>> users = {'momor', 'hamini', 'Justin'}
>>> 'Justin' in admins
True
>>> admins & users
set(['Justin'])
>>> admins | users
set(['hamini', 'caterpillar', 'Justin', 'momor'])
>>> admins - users
set(['caterpillar'])
>>> admins ^ users
set(['hamini', 'caterpillar', 'momor'])
>>> admins > users
False
>>> admins < users
False
>>>
```

Dict Type

- An object that maps keys to values.



```
>>> passwords = {'Justin' : 123456, 'caterpillar' : 933933}
>>> passwords['Justin']
123456
>>> passwords['Hamimi'] = 970221
>>> passwords
{'caterpillar': 933933, 'Hamimi': 970221, 'Justin': 123456}
>>> del passwords['caterpillar']
>>> passwords
{'Hamimi': 970221, 'Justin': 123456}
>>> passwords.items()
[('Hamimi', 970221), ('Justin', 123456)]
>>> passwords.keys()
['Hamimi', 'Justin']
>>> passwords.values()
[970221, 123456]
>>> passwords.get('openhome', '000000')
'000000'
>>> passwords['openhome']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'openhome'
>>>
```

A tuple

```
if 'openhome' in passwords:
    return passwords['openhome']
else:
    return '000000'
```

Tuple Type

- A tuple is like a list, yet it’s **immutable**.
- Shares common operations with lists.
  - In fact, sequences in Python (e.g. strings, lists, tuples, etc.) shares several features.
- Mutable or immutable? We’ll talk about it soon…
- (In Haskell - a statically-typed language - the types of elements in a tuple composes an unnamed type.)

Exercise 3

- Open a terminal and type `python`. What will you see in **the interactive shell** if you type the following commands?
  - `1 + 2`
  - `_`
  - `_ + 3`
  - `help()`
  - `len`
  - `keywords`
  - `quit`(or simply `q`)
  - `help(len)`
  - Ctrl + D

- After exiting the interactive shell, what will you see in the terminal if you type the following commands?
  - `python -h`
  - `python -c 'print "Hello! Python!"'`
  - `python -c 'help(len)'`
  - `python -c 'import this'`
- (Try anything you see from the previous slides about built-in types.)

`if`, `for`, `while` and `for` comprehensions

- `if..else` block

```
from sys import argv
if len(argv) != 1:
    print 'Hello, ' + argv[1]
else:
    print 'Hello, Guest'
```

Below is a block

Indentation is important.

- `if..else` expression, something like the ternary operator `?:` in C or Java.

```
from sys import argv
print 'Hello, ' + (argv[1] if len(argv) != 1 else 'Guest')
```

`for` and `while`

- Use `for in` to iterate a sequence.

```
numbers = [10, 20, 30]
squares = []
for number in numbers:
    squares.append(number ** 2)
print squares
```

- Use `while` for undetermined conditions.

```
print 'Enter two numbers...'
m = int(raw_input('Number 1: '))
n = int(raw_input('Number 2: '))
while n != 0:
   r = m % n
   m = n
   n = r
print 'GCD: {0}'.format(m)
```

`for` comprehensions

- With a list comprehension we can turn this:

```
numbers = [10, 20, 30]
squares = []
for number in numbers:
    squares.append(number ** 2)
print squares
```

- Into this:

```
numbers = [10, 20, 30]
print [number ** 2 for number in numbers]
```

- With a list comprehension we can turn this:

```
numbers = [11, 2, 45, 1, 6, 3, 7, 8, 9]
odd_numbers = []
for number in numbers:
    if number % 2 != 0:
        odd_numbers.append(number)
print odd_numbers
```

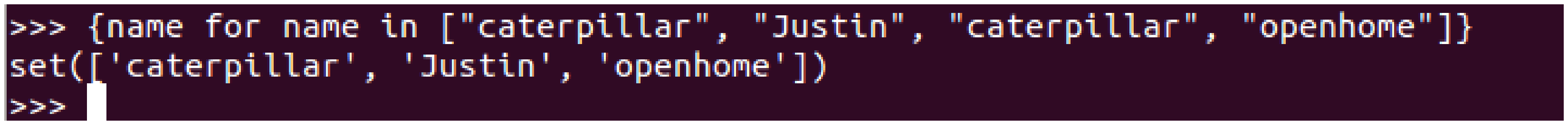
- Into this:

```
numbers = [11, 2, 45, 1, 6, 3, 7, 8, 9]
print [number for number in numbers if number % 2 != 0]
```

- Flatten a list of lists.

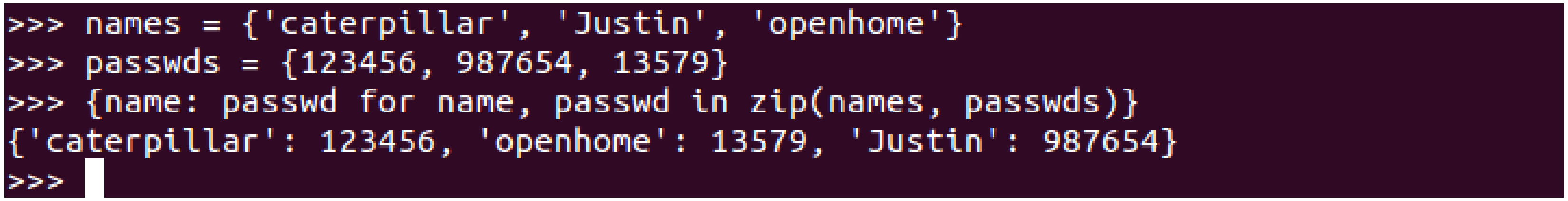
```
lts = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
print [ele for lt in lts for ele in lt]
```

- A set comprehension



```
>>> {name for name in ["caterpillar", "Justin", "caterpillar", "openhome"]}
set(['caterpillar', 'Justin', 'openhome'])
>>> 
```

- A dict comprehension



```
>>> names = {'caterpillar', 'Justin', 'openhome'}
>>> passwds = {123456, 987654, 13579}
>>> {name: passwd for name, passwd in zip(names, passwds)}
{'caterpillar': 123456, 'openhome': 13579, 'Justin': 987654}
>>> 
```

- (In Haskell, a set comprehension $S = \{2 \cdot x \mid x \in \mathbb{N},\ x \leq 10\}$ in mathematics can be written as [2 * x | x <- N, x <= 10] which looks similar to the set comprehension.)

Exercise 4

- Turn the following code into a single statement.

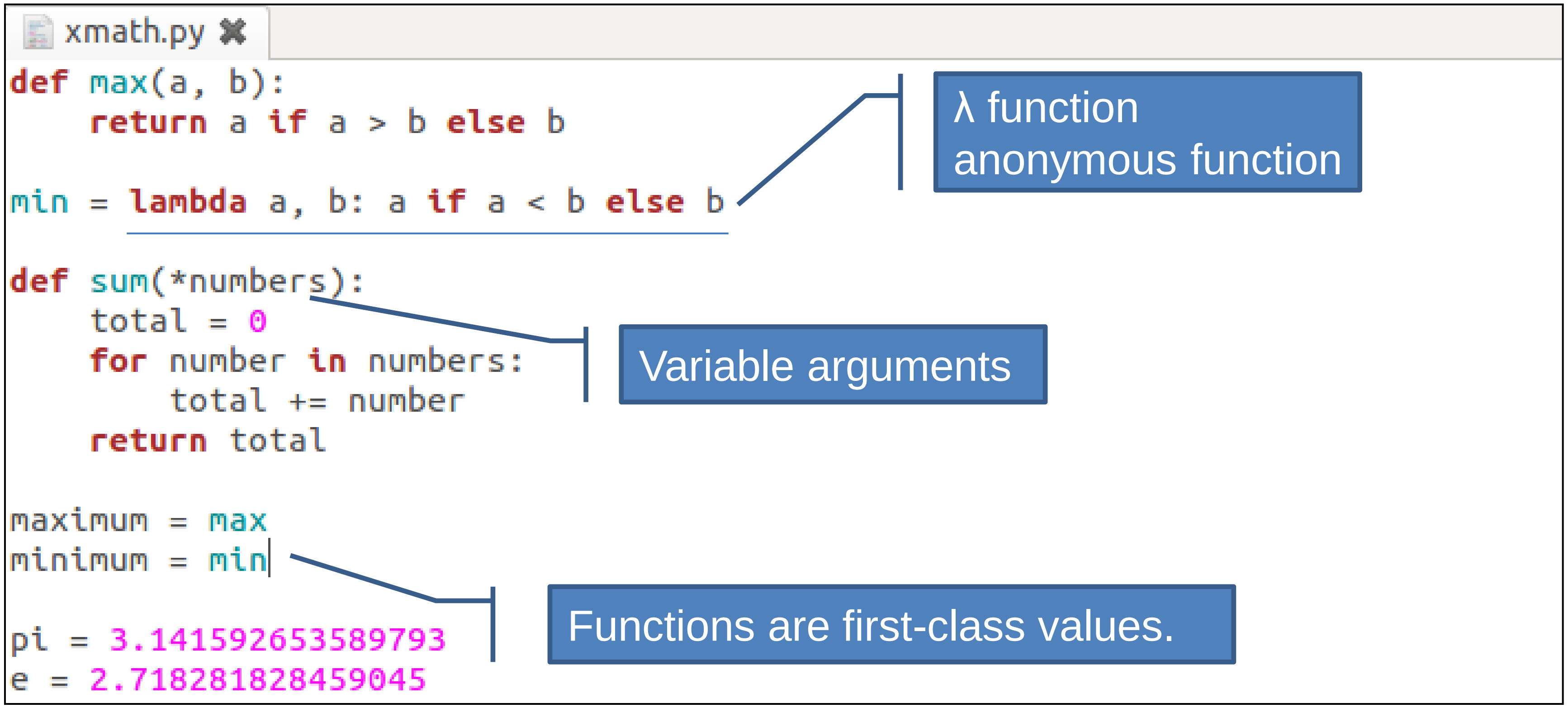
```
numbers = []
for number in range(20):
    numbers.append(str(number))
print ", ".join(numbers)
```

- (Here's a problem that combines tuple and list comprehensions: which right triangle that has integers for all sides and all sides equal to or smaller than 10 has a perimeter of 24?)

Functions, Modules, Classes and Packages

- In Python, everything is an object.
  - ***Does Python impose object-oriented programming as the main programming paradigm?***
- Points about structuring your program.
  - Encapsulation and separation of abstraction layers.
  - State of an object.
  - Namespace
  - Physical structures of your resources, such as source files, packages, etc.

Functions



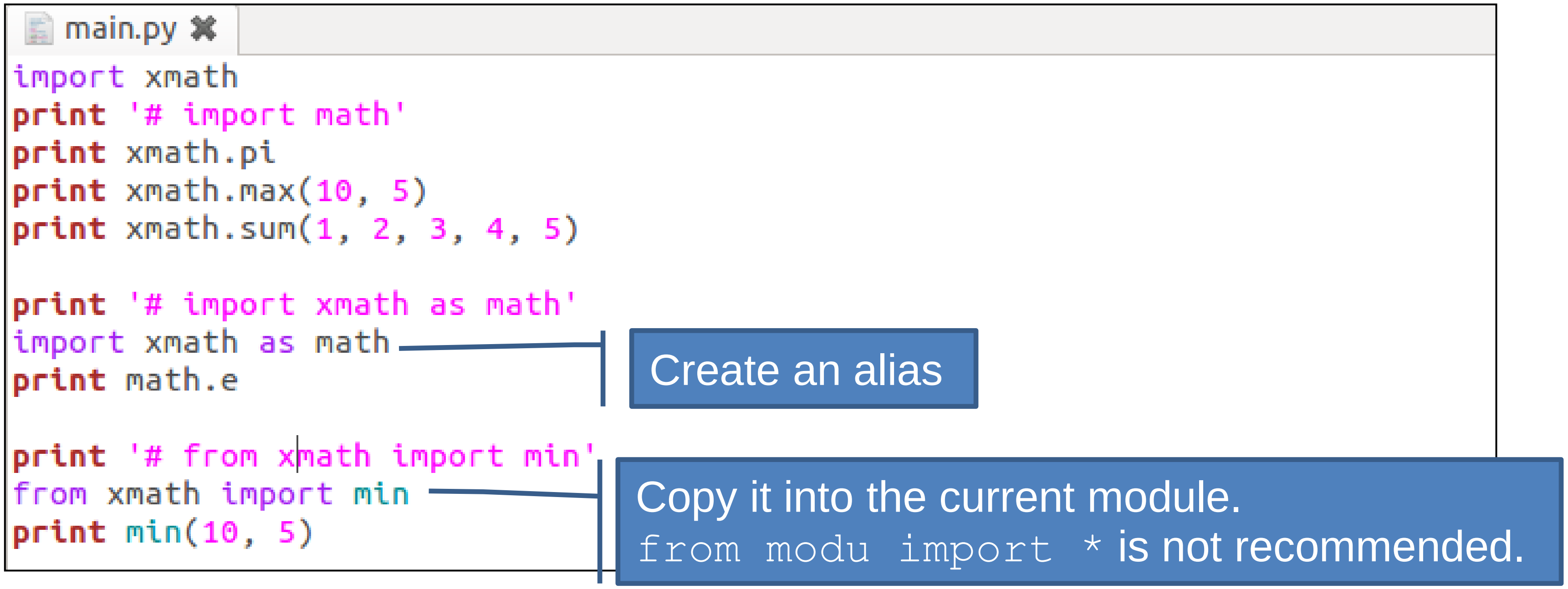
xmath.py

```
def max(a, b):
    return a if a > b else b

min = lambda a, b: a if a < b else b

def sum(*numbers):
    total = 0
    for number in numbers:
        total += number
    return total

maximum = max
minimum = min

pi = 3.141592653589793
e = 2.718281828459045
```

Modules

- What's the best way to organize functions in the previous slide?
- Modules are one of the main abstraction layers available and probably the most natural one.
  - A file named modu.py creates a module `modu`.
  - The `import modu` statement will look for *modu.py* in the same. If it isn't found, the Python interpreter will search for modu.py in the `sys.path` recursively; or raise an ImportError exception if it isn't found.

- **A module provides a namespace.** The module's variables, functions, and classes will be available to the caller through the module's namespace
- `import`, `import as`, `from import` are statements.

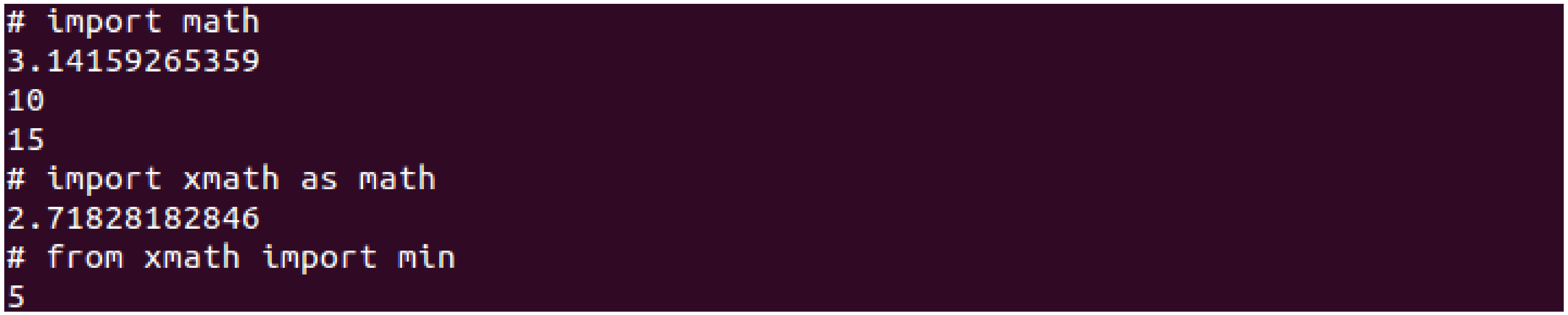


main.py

```
import xmath
print '# import math'
print xmath.pi
print xmath.max(10, 5)
print xmath.sum(1, 2, 3, 4, 5)

print '# import xmath as math'
import xmath as math
print math.e

print '# from xmath import min'
from xmath import min
print min(10, 5)
```

Create an alias

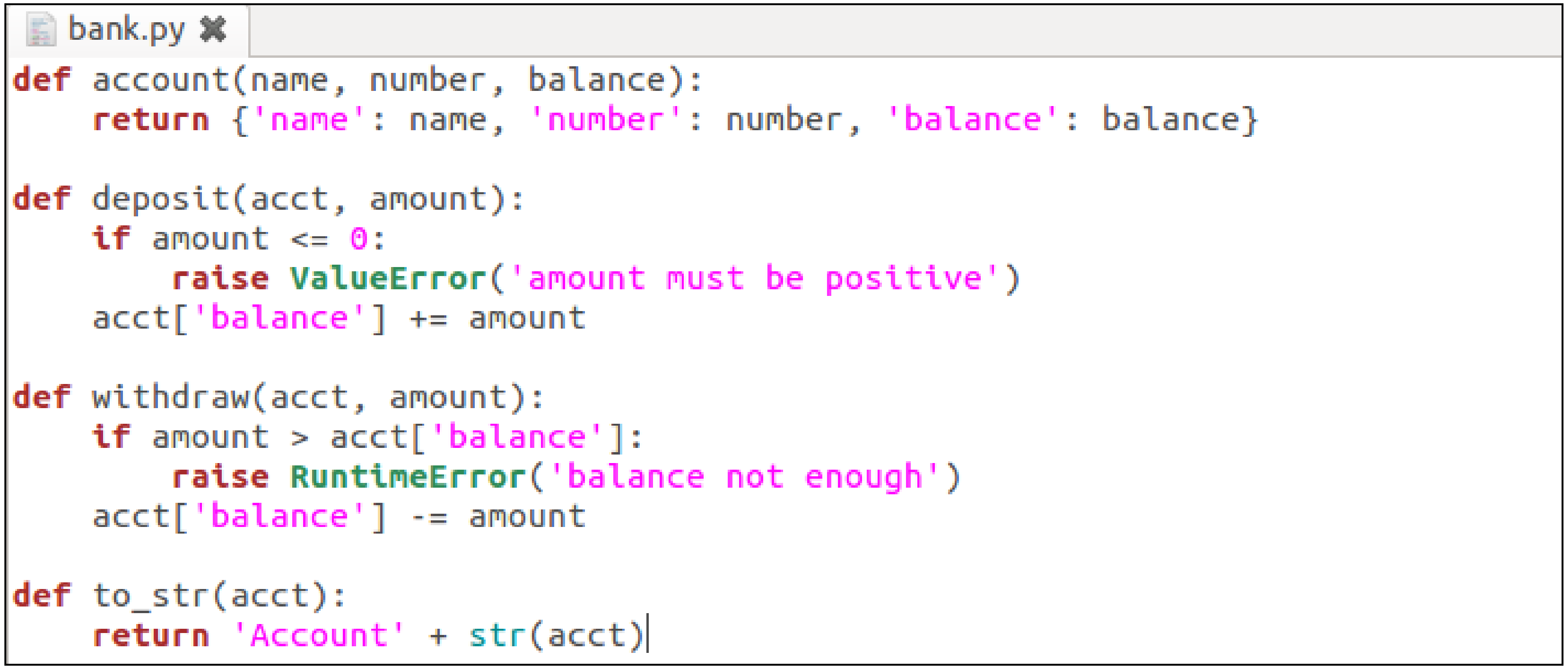
Copy it into the current module.
`from modu import *` is not recommended.



```
# import math
3.14159265359
10
15
# import xmath as math
2.71828182846
# from xmath import min
5
```

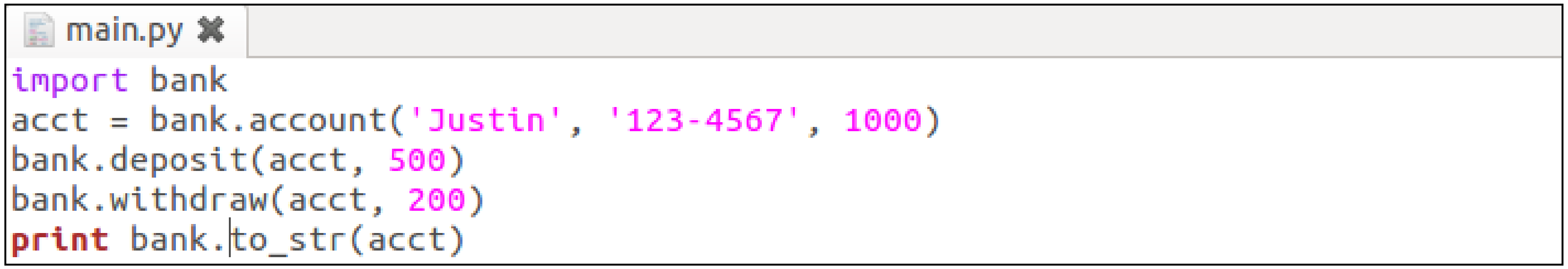
Classes

- Well, where's the playground for classes?
  - When we want to glue together some **state** and some functionality.



bank.py

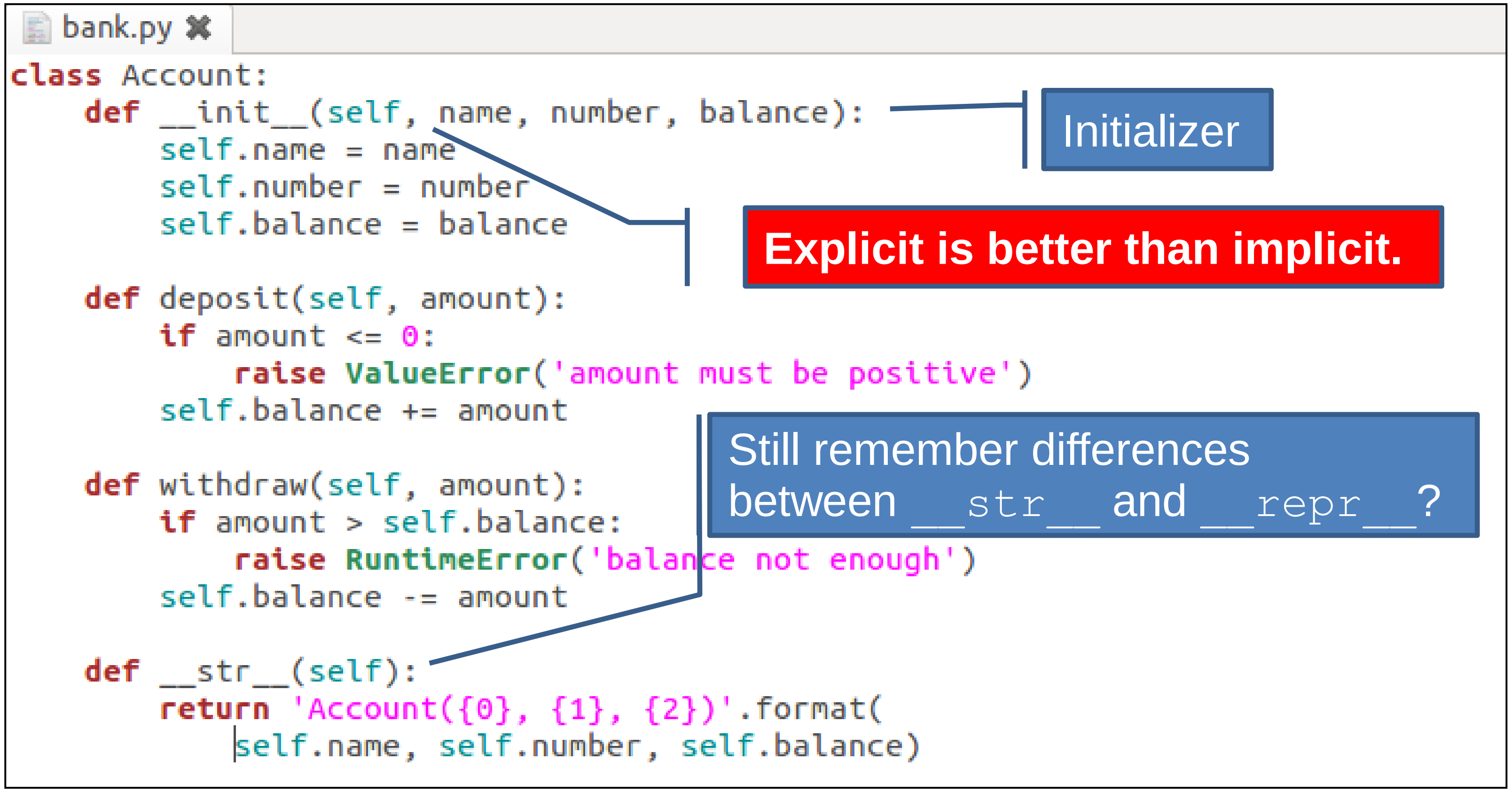
```python
def account(name, number, balance):
    return {'name': name, 'number': number, 'balance': balance}

def deposit(acct, amount):
    if amount <= 0:
        raise ValueError('amount must be positive')
    acct['balance'] += amount

def withdraw(acct, amount):
    if amount > acct['balance']:
        raise RuntimeError('balance not enough')
    acct['balance'] -= amount

def to_str(acct):
    return 'Account' + str(acct)
```



main.py

```python
import bank
acct = bank.account('Justin', '123-4567', 1000)
bank.deposit(acct, 500)
bank.withdraw(acct, 200)
print bank.to_str(acct)
```

- **OOP is considering usability more than reusability.**



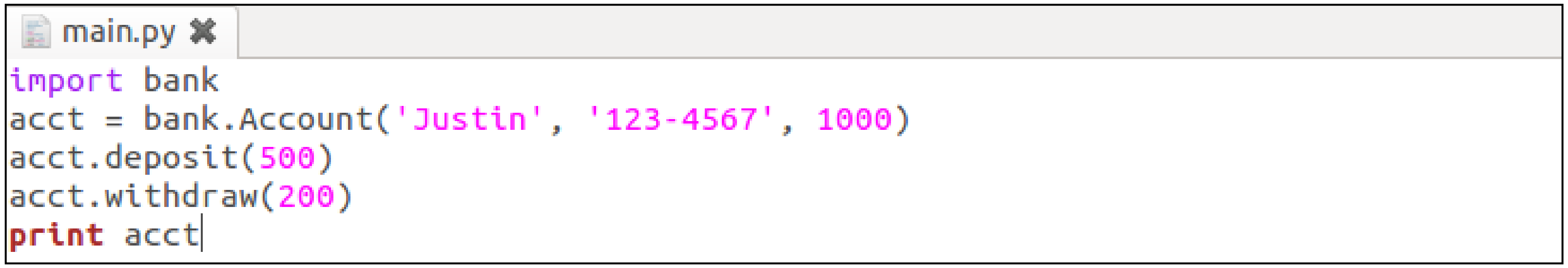
bank.py

```python
class Account:
    def __init__(self, name, number, balance):
        self.name = name
        self.number = number
        self.balance = balance

    def deposit(self, amount):
        if amount <= 0:
            raise ValueError('amount must be positive')
        self.balance += amount

    def withdraw(self, amount):
        if amount > self.balance:
            raise RuntimeError('balance not enough')
        self.balance -= amount

    def __str__(self):
        return 'Account({0}, {1}, {2})'.format(
            self.name, self.number, self.balance)
```

Initializer

Explicit is better than implicit.

Still remember differences between `__str__` and `__repr__`?



main.py

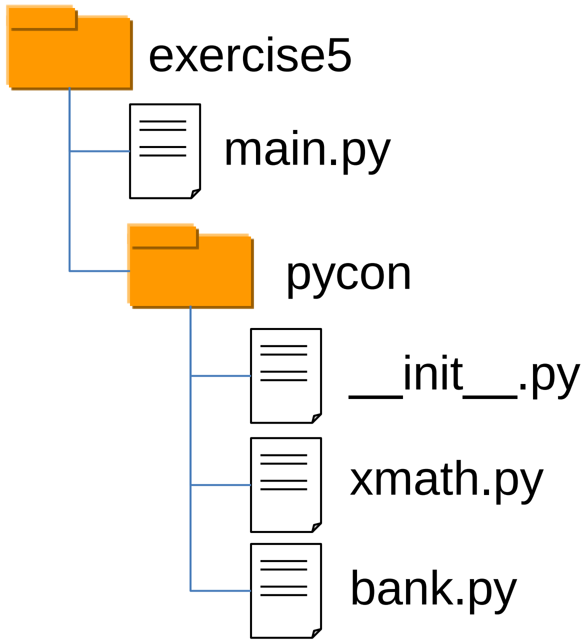
```python
import bank
acct = bank.Account('Justin', '123-4567', 1000)
acct.deposit(500)
acct.withdraw(200)
print acct
```

Packages

- Any directory with an __init__.py file - used to gather all package-wide definitions - is considered a package.
- `import pack.modu` will looks for a file **modu.py** in the directory **pack**.
  - This statement will look for an __init__.py file in the directory pack, execute all of its top-level statements.
  - Then it will look for a file pack/modu.py and execute all of its top-level statements.
  - After these operations, any variable, function, or class defined in modu.py is available in the `pack.modu` namespace.

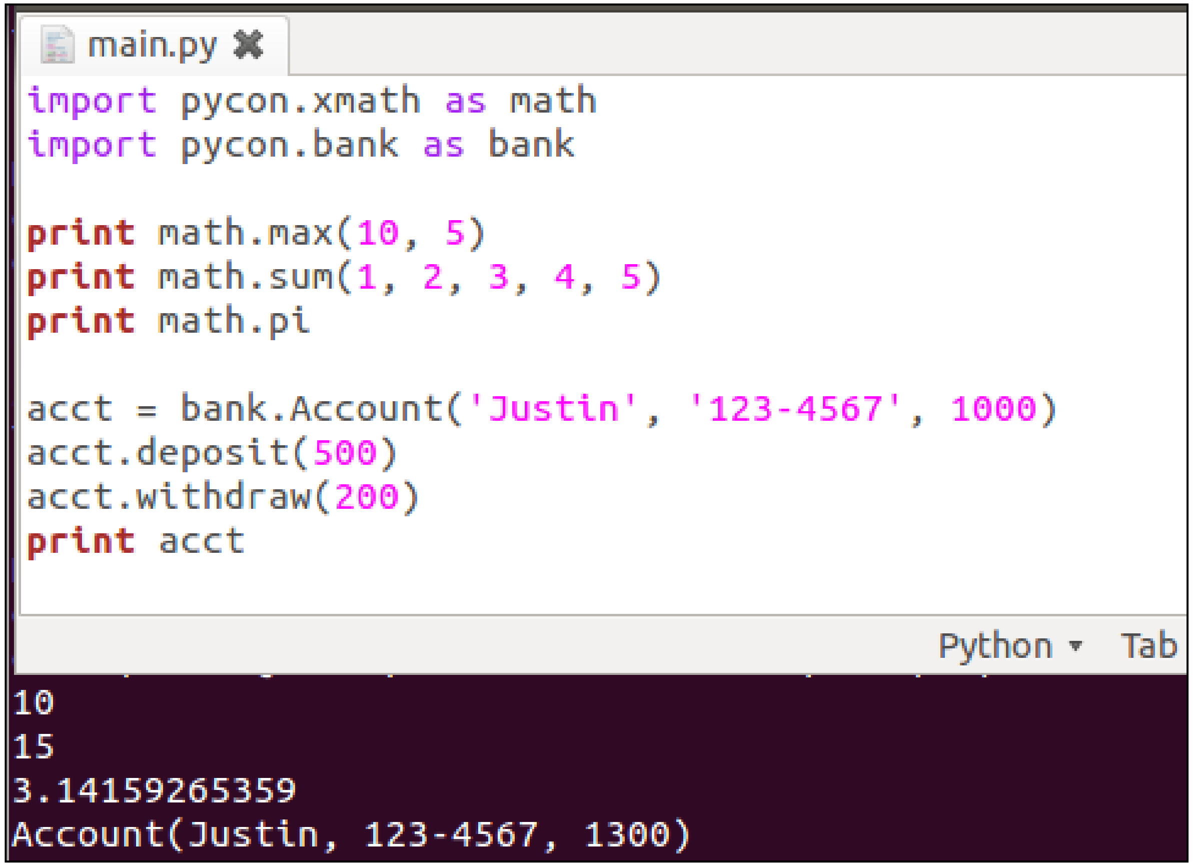
Exercise 5

- There's a quick and dirty **main.py** located in the **/exercises/exercise5** of the lab file. Use modules, classes and packages learned in the previous slides to structure them as follow:



What You Should See?

- Basically, you should have the following main.py and run it correctly.



```
import pycon.xmath as math
import pycon.bank as bank

print math.max(10, 5)
print math.sum(1, 2, 3, 4, 5)
print math.pi

acct = bank.Account('Justin', '123-4567', 1000)
acct.deposit(500)
acct.withdraw(200)
print acct
```

```
10
15
3.14159265359
Account(Justin, 123-4567, 1300)
```

References

- String Type
  - http://docs.python.org/2/reference/datamodel.html#object.__repr__
  - http://docs.python.org/py3k/library/stdtypes.html#old-string-formatting
  - http://docs.python.org/py3k/library/string.html#string-formatting
- List, Set, Dict, Tuple Types
  - http://caterpillar.onlyfun.net/Gossip/Python/ListType.html
  - http://caterpillar.onlyfun.net/Gossip/Python/SetType.html
  - http://caterpillar.onlyfun.net/Gossip/Python/DictionaryType.html
  - http://caterpillar.onlyfun.net/Gossip/Python/TupleType.html
- Functions, Modules, Classes and Packages
  - http://caterpillar.onlyfun.net/Gossip/Python/Class.html
  - http://caterpillar.onlyfun.net/Gossip/Python/Class.html
  - http://docs.python-guide.org/en/latest/writing/structure/
- Short Cuts
  - http://maxburstein.com/blog/python-shortcuts-for-the-python-beginner/

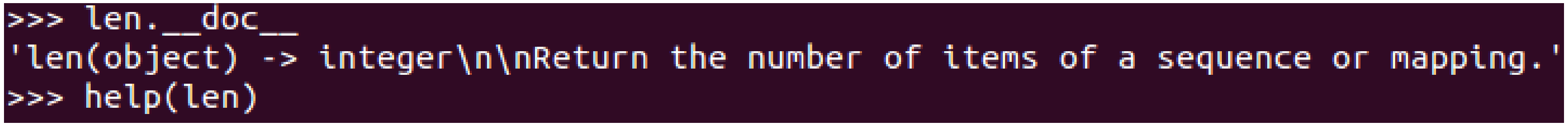
The Community

- BDFL
  - **Guido van Rossum** (www.python.org/~guido)
  - The creator of Python, is often referred to as the **Benevolent Dictator For Life**.
- PSF
  - **Python Software Foundation** (www.python.org/psf)
  - Its mission is to promote, protect, and advance the Python programming language, and to support and facilitate the growth of a diverse and international community of Python programmers.
  - A 501(c)(3) non-profit corporation that holds the intellectual property rights behind the Python programming language.

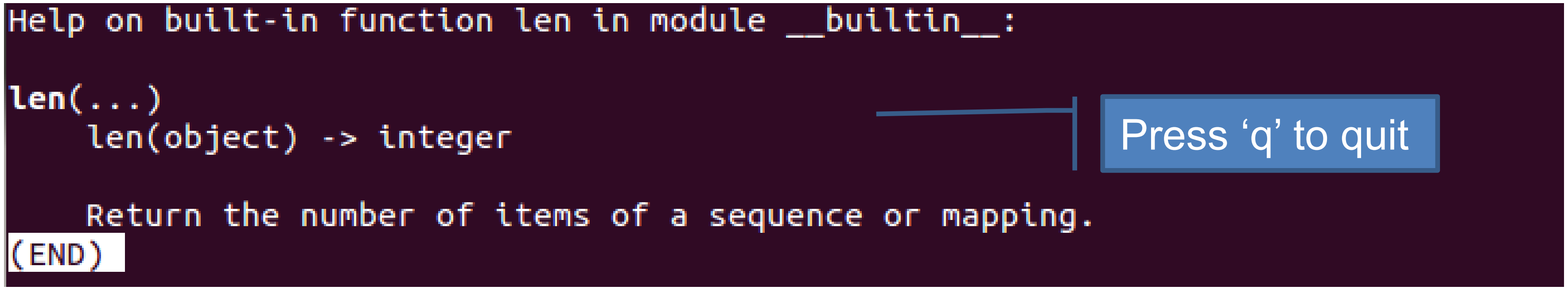
- PEPs
  - **Python Enhancement Proposals** (www.python.org/dev/peps)
  - Describes changes to Python itself, or the standards around it.
  - Notable PEPs
    - PEP 1 -- PEP Purpose and Guidelines.
    - PEP 8 -- Style Guide for Python Code
    - PEP 20 -- The Zen of Python
    - PEP 257 -- Docstring Conventions
- PyCon
  - **Python Conference** (www.pycon.org)
  - PyCon Taiwan (tw.pycon.org)
- PIGgies
  - **Python User Groups** (wiki.python.org/moin/LocalUserGroups)
  - Taiwan Python User Group (wiki.python.org.tw)

Documentation

- What happens if you type `len.__doc__` in the interactive shell?
- Remember `help`? What's the relationship between `help(len)` and `len.__doc__`?
- Where's `len.__doc__` from?



```
>>> len.__doc__
'len(object) -> integer\n\nReturn the number of items of a sequence or mapping.'
>>> help(len)
```



```
Help on built-in function len in module __builtin__:

len(...)
    len(object) -> integer

    Return the number of items of a sequence or mapping.
(END)
```

DocStrings

- Type the following code in the interactive shell.

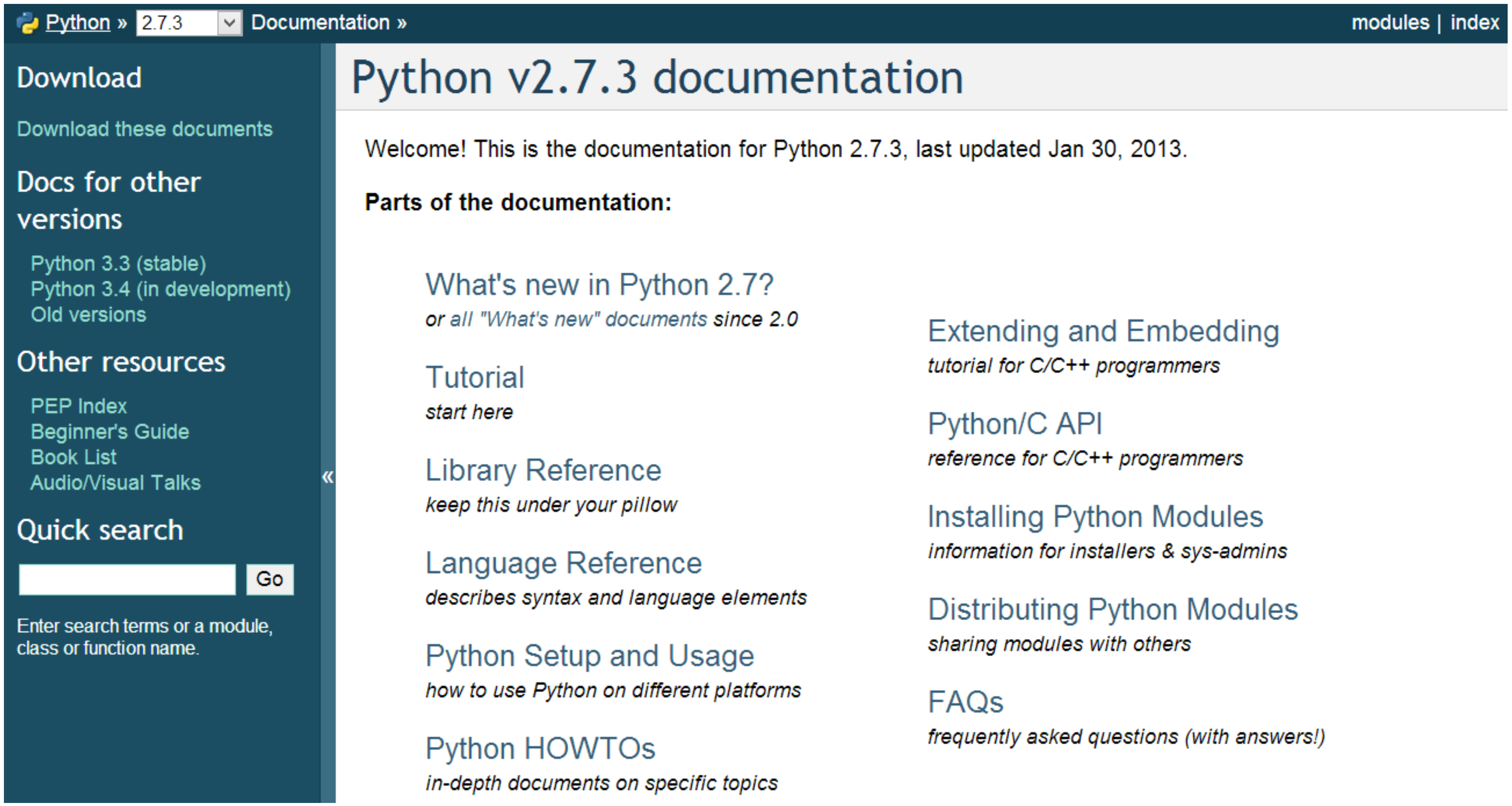
```
def max(a, b):
    '''max(a, b) -> value

       With two arguments, return the largest argument.'''
    return a if a > b else b
```

- Type `max.__doc__` in the interactive shell.
- Type `help(max)` in the interactive shell.
- You’ll know what DocStrings are.
- Remember to read **PEP 257** if you want to comply with **DocString Conventions**.

Official Documentation

- docs.python.org

Python Module Index



Download

Download these documents

Docs for other versions

Python 3.3 (stable)
Python 3.4 (in development)
Old versions

Other resources

PEP Index
Beginner's Guide
Book List
Audio/Visual Talks

Quick search

Go

Enter search terms or a module, class or function name.

Python v2.7.3 documentation

Welcome! This is the documentation for Python 2.7.3, last updated Jan 30, 2013.

Parts of the documentation:

What's new in Python 2.7?
or all "What's new" documents since 2.0

Tutorial
start here

Library Reference
keep this under your pillow

Language Reference
describes syntax and language elements

Python Setup and Usage
how to use Python on different platforms

Python HOWTOs
in-depth documents on specific topics

Extending and Embedding
tutorial for C/C++ programmers

Python/C API
reference for C/C++ programmers

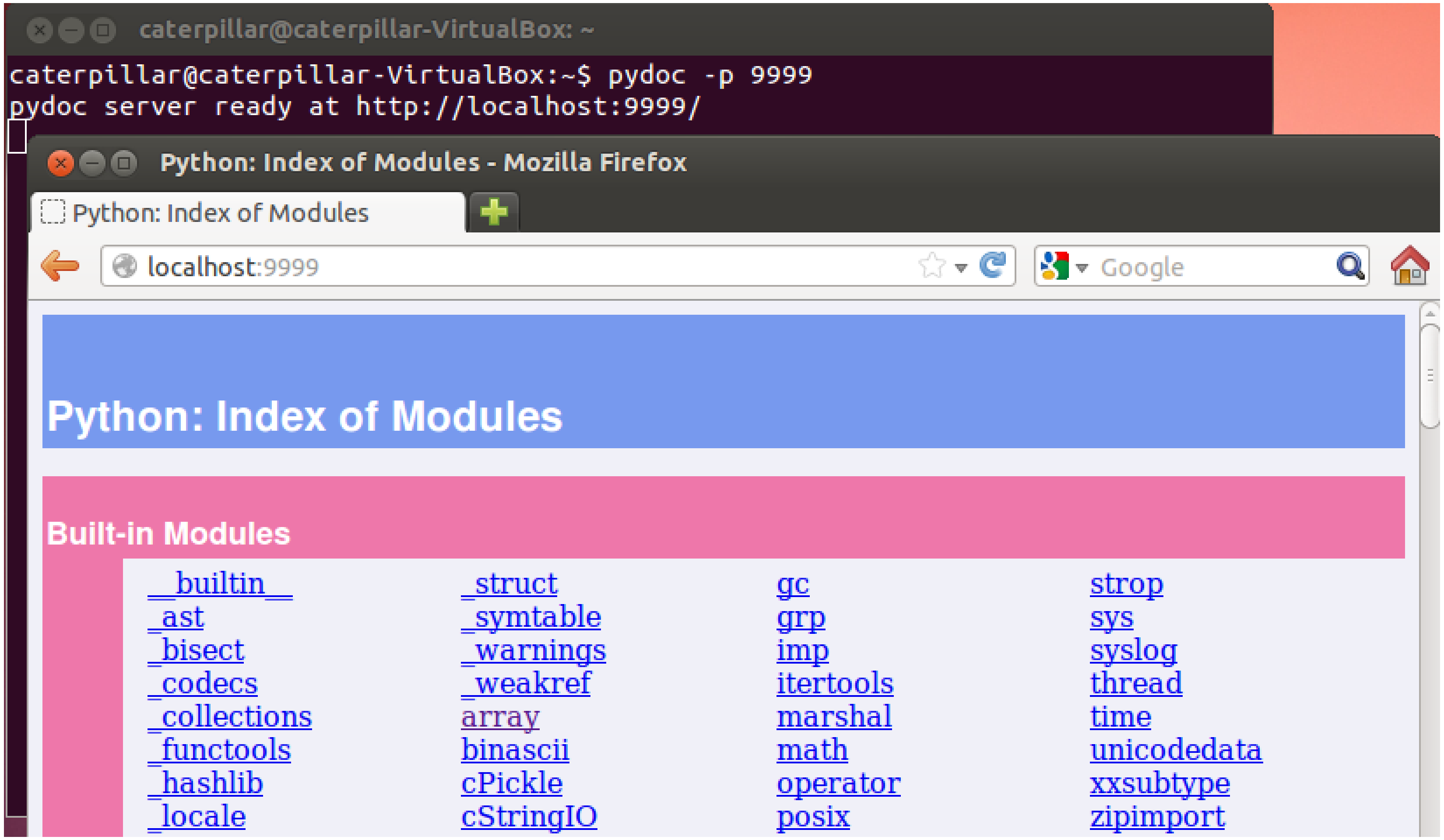
Installing Python Modules
information for installers & sys-admins

Distributing Python Modules
sharing modules with others

FAQs
frequently asked questions (with answers!)

PyDoc

- The pydoc module automatically generates documentation from Python modules.



```
caterpillar@caterpillar-VirtualBox:~$ pydoc -p 9999
pydoc server ready at http://localhost:9999/
```

caterpillar@caterpillar-VirtualBox: ~
Python: Index of Modules - Mozilla Firefox
Python: Index of Modules
localhost:9999
Google

Python: Index of Modules

Built-in Modules

__builtin__	_struct	gc	strop
_ast	_symtable	grp	sys
_bisect	_warnings	imp	syslog
_codecs	_weakref	itertools	thread
_collections	array	marshal	time
_functools	binascii	math	unicodedata
_hashlib	cPickle	operator	xxsubtype
_locale	cStringIO	posix	zipimport

EpyDoc

- Looks for something like JavaDoc?
- epydoc.sourceforge.net

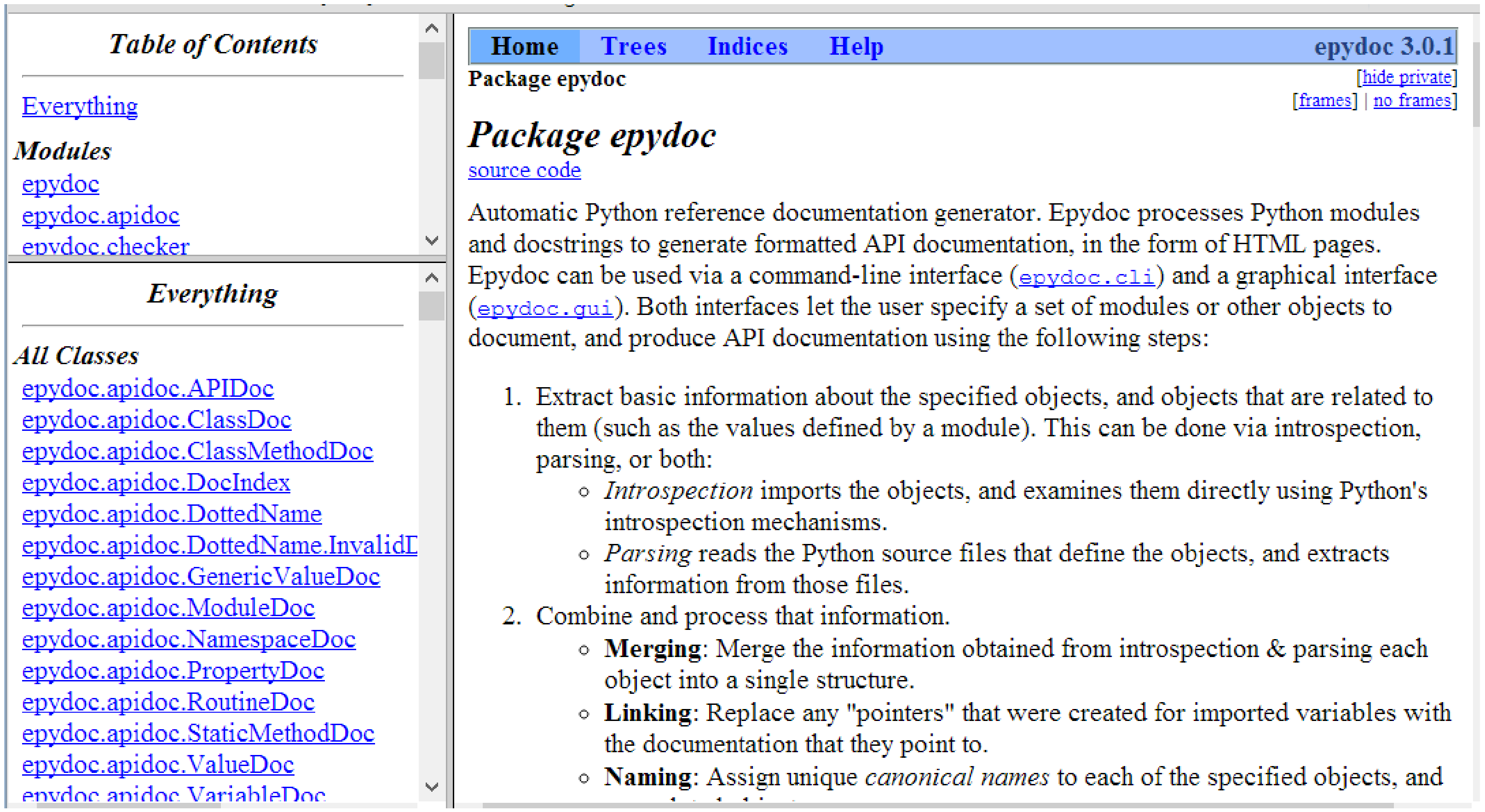


Table of Contents

Everything

Modules

epydoc
epydoc.apidoc
epydoc.checker

Everything

All Classes

epydoc.apidoc.APIDoc
epydoc.apidoc.ClassDoc
epydoc.apidoc.ClassMethodDoc
epydoc.apidoc.DocIndex
epydoc.apidoc.DottedName
epydoc.apidoc.DottedName.InvalidD
epydoc.apidoc.GenericValueDoc
epydoc.apidoc.ModuleDoc
epydoc.apidoc.NamespaceDoc
epydoc.apidoc.PropertyDoc
epydoc.apidoc.RoutineDoc
epydoc.apidoc.StaticMethodDoc
epydoc.apidoc.ValueDoc
epydoc.apidoc.VariableDoc

Home **Trees** **Indices** **Help** **epydoc 3.0.1**

Package epydoc

[hide private]
[frames] | no frames]

Package epydoc

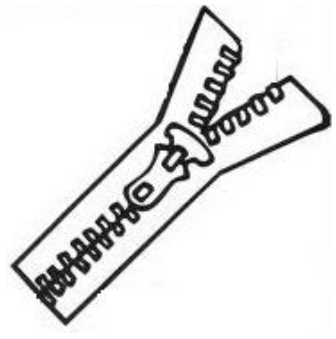
source code

Automatic Python reference documentation generator. Epydoc processes Python modules and docstrings to generate formatted API documentation, in the form of HTML pages. Epydoc can be used via a command-line interface (`epydoc.cli`) and a graphical interface (`epydoc.gui`). Both interfaces let the user specify a set of modules or other objects to document, and produce API documentation using the following steps:

1. Extract basic information about the specified objects, and objects that are related to them (such as the values defined by a module). This can be done via introspection, parsing, or both:
   - *Introspection* imports the objects, and examines them directly using Python's introspection mechanisms.
   - *Parsing* reads the Python source files that define the objects, and extracts information from those files.
2. Combine and process that information.
   - **Merging**: Merge the information obtained from introspection & parsing each object into a single structure.
   - **Linking**: Replace any "pointers" that were created for imported variables with the documentation that they point to.
   - **Naming**: Assign unique *canonical names* to each of the specified objects, and

Data Management Functions

- Built-in Functions (located in the `__builtin__` module)
  - `range(start, stop[, step])`
  - `zip([iterable, ...])`
  - `enumerate(sequence, start=0)`
  - `reduce(function, iterable[, initializer])`



Exercise 6

- How to iterate through a list with an index? For examples, given a list `names = ['Justin', 'caterpillar', 'openhome']`, print the followings.

```
0, Justin
1, caterpillar
2, openhome
```

- Hints:
  - 1. Fill in the blanks with proper codes.

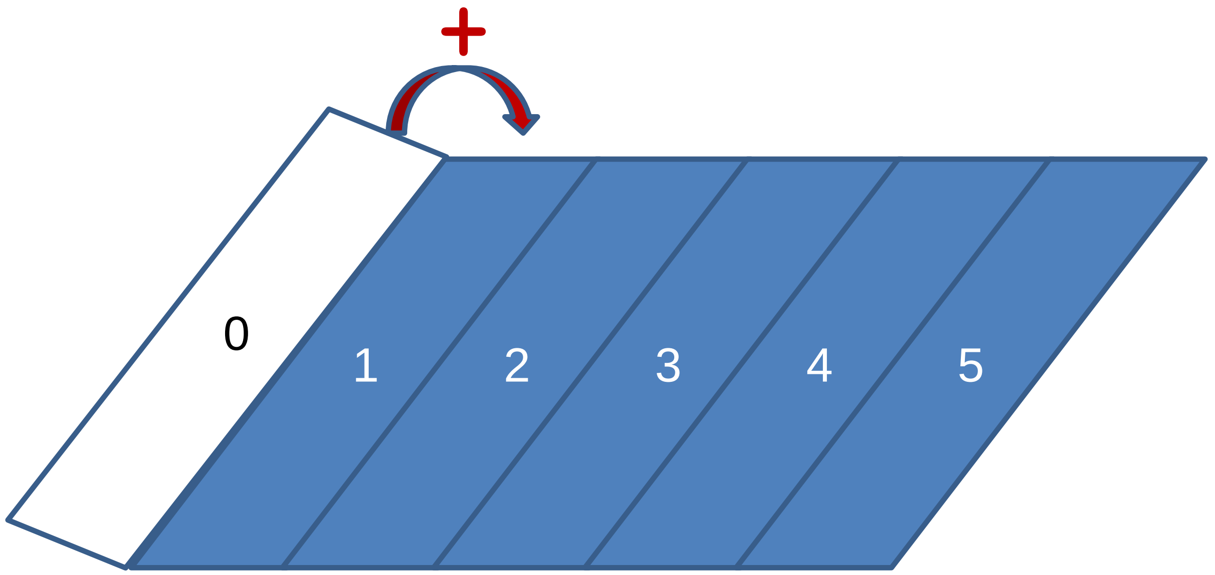
```
names = ['Justin', 'caterpillar', 'openhome']
for ______ in ______:
    print '{0}, {1}'.format(______)
```

  - 2. Look up documentations about `range`, `zip` and `enumerate`.

`reduce`

- Sometimes, it's called `foldLeft`.

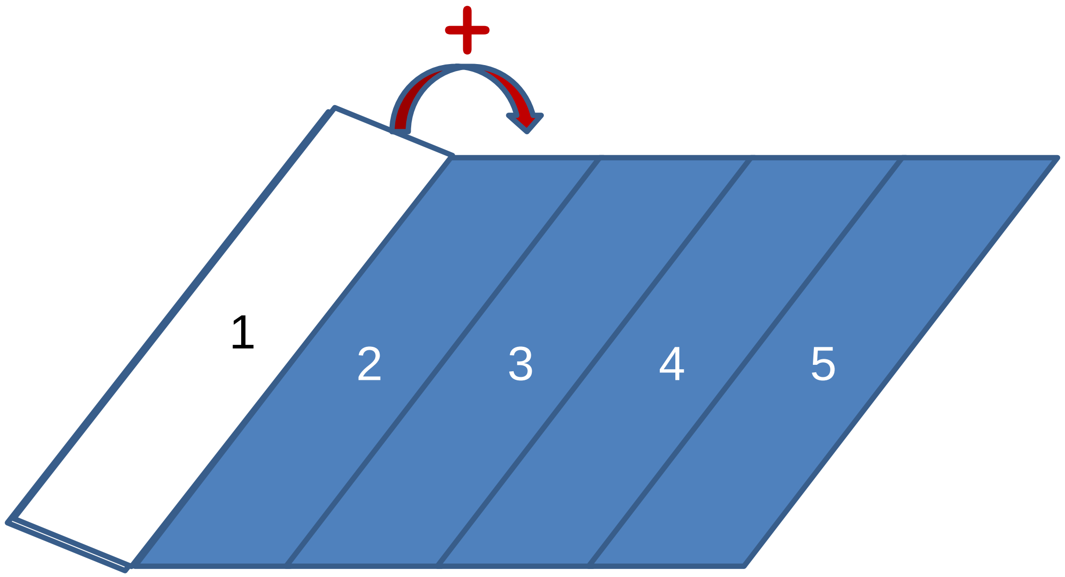
```
reduce(lambda acct, elem: acct + elem, [1, 2, 3, 4, 5], 0)
```



`reduce`

- Sometimes, it's called `foldLeft`.

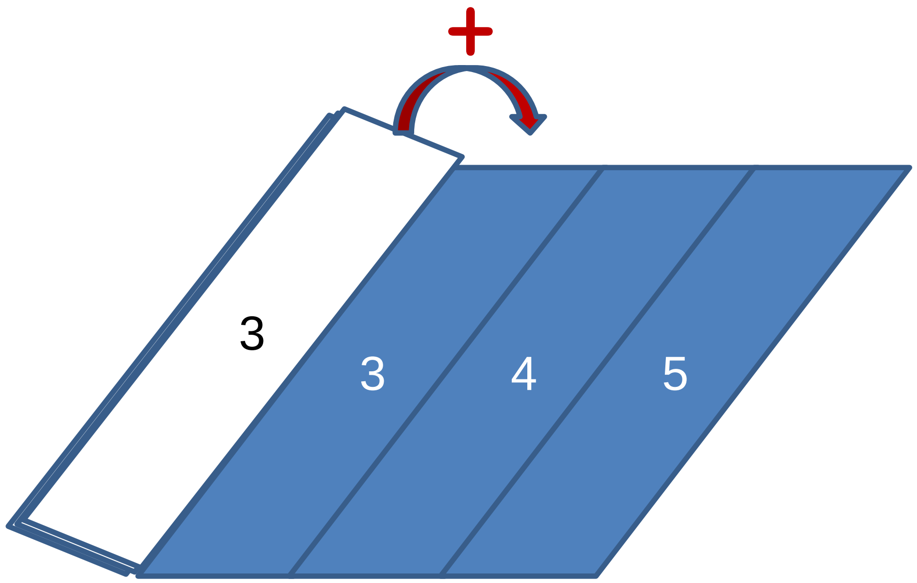
```
reduce(lambda acct, elem: acct + elem, [1, 2, 3, 4, 5], 0)
```



reduce

- Sometimes, it's called `foldLeft`.

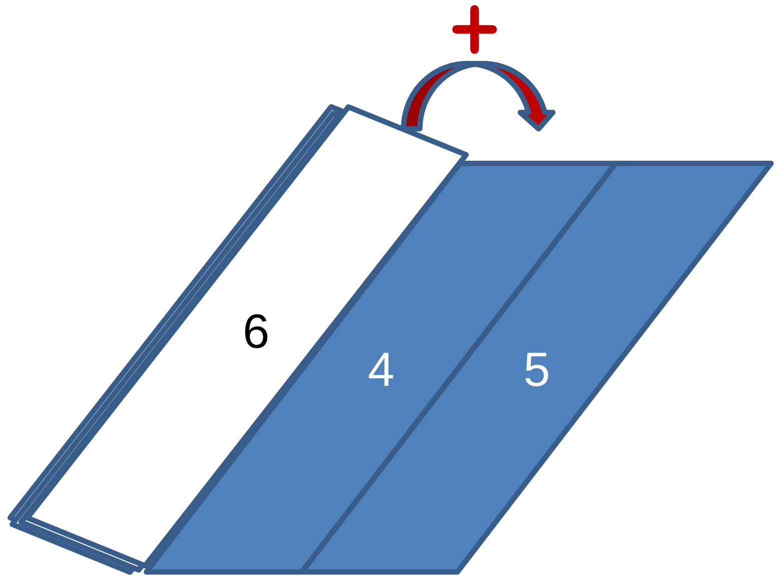
```
reduce(lambda acct, elem: acct + elem, [1, 2, 3, 4, 5], 0)
```



reduce

- Sometimes, it's called `foldLeft`.

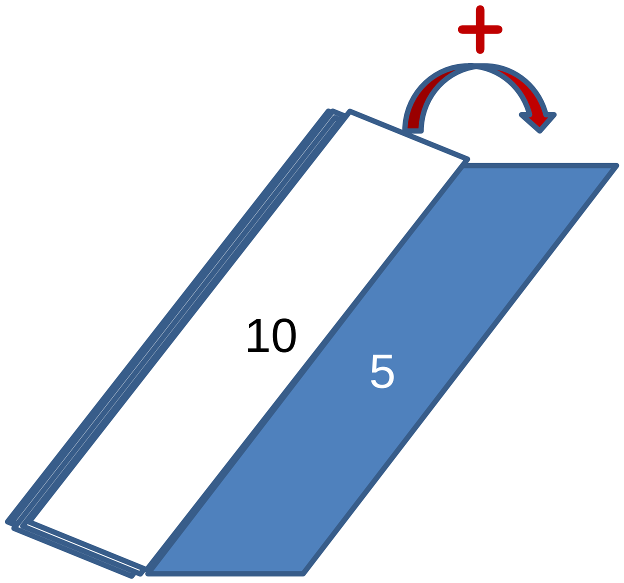
```
reduce(lambda acct, elem: acct + elem, [1, 2, 3, 4, 5], 0)
```



`reduce`

- Sometimes, it's called `foldLeft`.

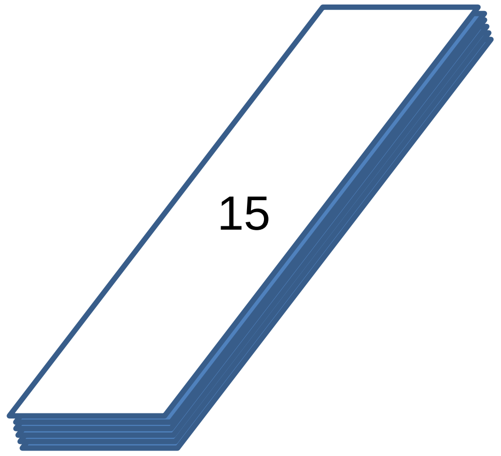
```
reduce(lambda acct, elem: acct + elem, [1, 2, 3, 4, 5], 0)
```



`reduce`

- Sometimes, it's called `foldLeft`.

```
reduce(lambda acct, elem: acct + elem, [1, 2, 3, 4, 5], 0)
```



`reduce`

- `reduce` is a really versatile function that can be used in millions of different ways.
- Once you want to calculate something from a list, consider using `reduce` instead of a `for` loop.

Exercise 7

- Use **`reduce`** and **list comprehensions** to revise the following code (avaliable in lab/exercises/exercise7/main.py).

```
def ascending(a, b): return a - b
def descending(a, b): return -ascending(a, b)
# selection sort
def sorted(xs, compare = ascending):
    return [] if not xs else __select(xs, compare)

def __select(xs, compare):
    selected = xs[0]
    for elem in xs[1:]:
        if compare(elem, selected) < 0:
            selected = elem

    remain = []
    selected_list = []
    for elem in xs:
        if elem != selected:
            remain.append(elem)
        else:
            selected_list.append(elem)

    return xs if not remain else selected_list + __select(remain, compare)

print sorted([2, 1, 3, 6, 5])
print sorted([2, 1, 3, 6, 5], descending)
```

Persistence

- Object serialization
  - `marshal`, `pickle`, `cPickle`
- DBM (Database Manager)
  - Simple “database” interface. Dbm objects behave like mappings (dictionaries) , except that keys and values are always strings.
- `shelve`
  - A “shelf” is a persistent, dictionary-like object. The values can be essentially arbitrary Python objects.
- DB-API 2.0 (PEP 249)
- Object-Relational Mapping (3rd-party libraries)
  - SQLAlchemy (www.sqlalchemy.org)
  - SQLObject (www.sqlobject.org)

`marshal`, `pickle`, `cPickle`

- A more primitive serialization module is `marshal`. It exists primarily to support Python’s .pyc files.
- In general, `pickle` should always be the preferred way to serialize Python objects.
  - It keeps track of the objects it has already serialized, so that later references to the same object won’t be serialized again.
  - It can serialize user-defined classes and their instances.
  - Its serialization format is guaranteed to be backwards compatible across Python releases.
- `cPickle` is written in C, so it can be up to 1000 times faster than pickle.

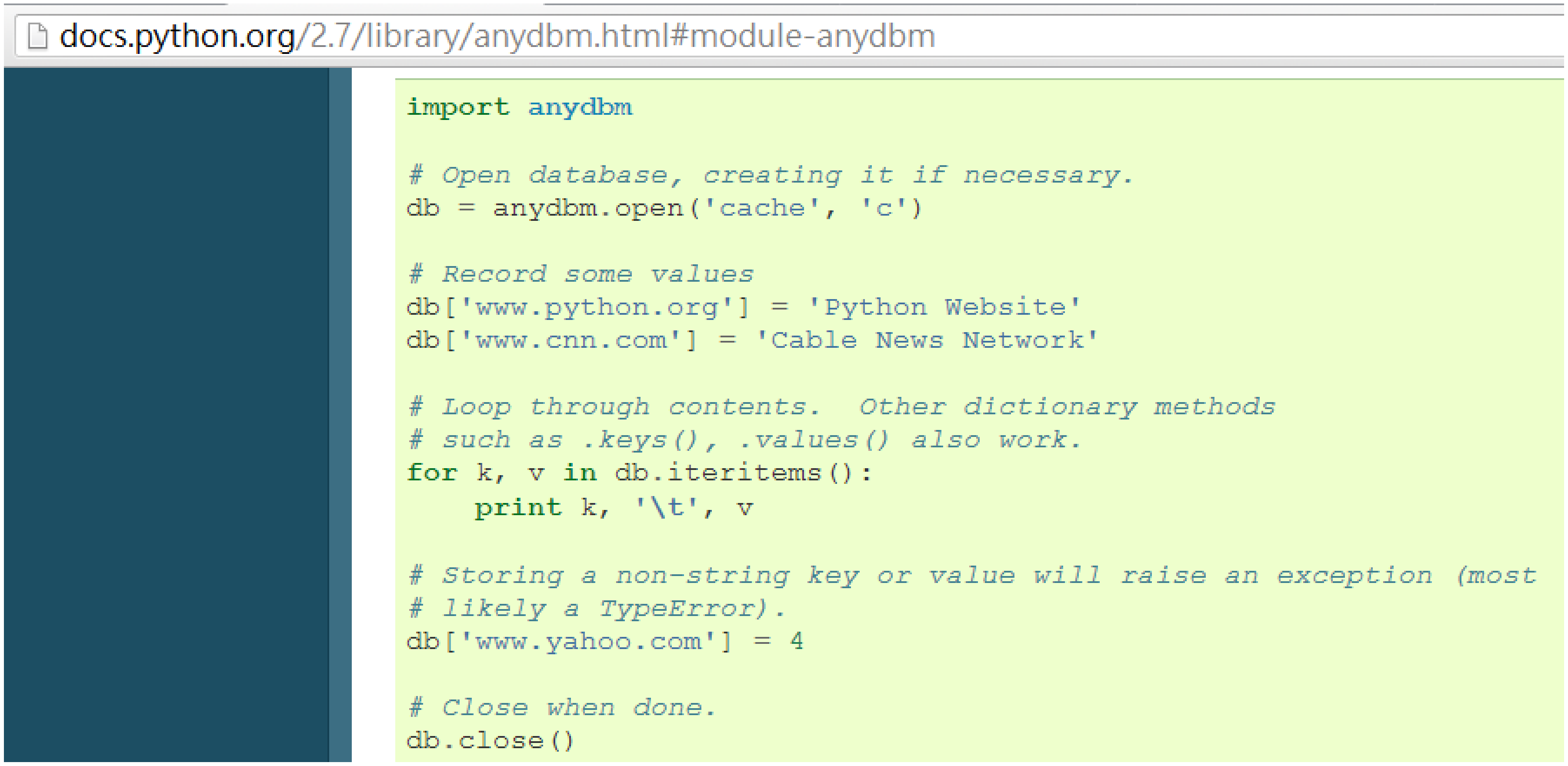
pickle

```
class DVD:
    def __init__(self, title, year=None,
        duration=None, director_id=None):
        self.title = title
        self.year = year
        self.duration = duration
        self.director_id = director_id
        self.filename = self.title.replace(' ', '_') + '.pkl'

    def check_filename(self, filename):
        if filename is not None:
            self.filename = filename
```

```
def save(self, filename=None):
    self.check_filename(filename)
    fh = None
    try:
        data = (self.title, self.year,
                self.duration, self.director_id)
        fh = open(self.filename, 'wb')
        pickle.dump(data, fh)
    except (EnvironmentError, pickle.PicklingError) as err:
        raise SaveError(str(err))
    finally:
        if fh is not None:
            fh.close()

def load(self, filename=None):
    self.check_filename(filename)
    fh = None
    try:
        fh = open(self.filename, 'rb')
        data = pickle.load(fh)
        (self.title, self.year,
         self.duration, self.director_id) = data
    except (EnvironmentError, pickle.PicklingError) as err:
        raise LoadError(str(err))
    finally:
        ...
```

DBM

- The `dbm` module provides an interface to the Unix “(n)dbm” library.



docs.python.org/2.7/library/anydbm.html#module-anydbm

```
import anydbm

# Open database, creating it if necessary.
db = anydbm.open('cache', 'c')

# Record some values
db['www.python.org'] = 'Python Website'
db['www.cnn.com'] = 'Cable News Network'

# Loop through contents.  Other dictionary methods
# such as .keys(), .values() also work.
for k, v in db.iteritems():
    print k, '\t', v

# Storing a non-string key or value will raise an exception (most
# likely a TypeError).
db['www.yahoo.com'] = 4

# Close when done.
db.close()
```

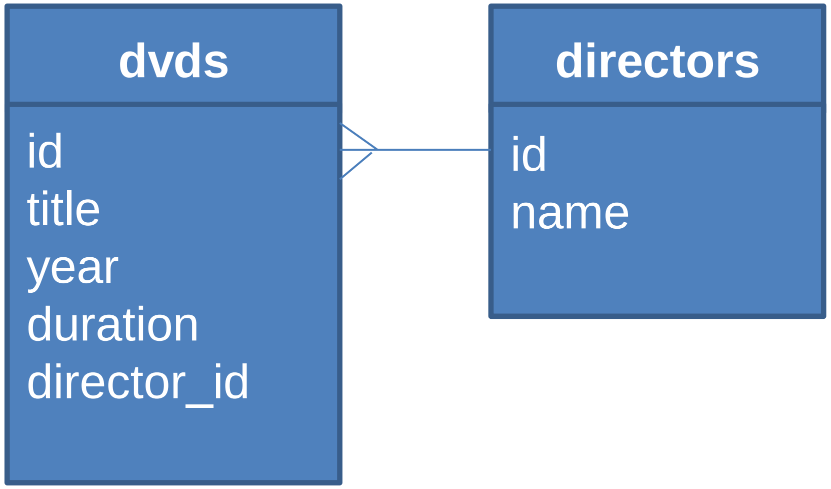
shelve

- A “shelf” is a persistent, dictionary-like object. The difference with “dbm” databases is that the values (not the keys!) in a shelf can be anything that the pickle module can handle.

```
class DvdDao:
    def __init__(self, shelve_name):
        self.shelve_name = shelve_name

    def save(self, dvd):
        shelve_db = None
        try:
            shelve_db = shelve.open(self.shelve_name)
            shelve_db[dvd.title] = (dvd.year,
                dvd.duration, dvd.director_id)
            shelve_db.sync()
        finally:
            if shelve_db is not None:
                shelve_db.close()
    def all(self):
        shelve_db = None
        try:
            shelve_db = shelve.open(self.shelve_name)
            return [DVD(title, *shelve_db[title])
                    for title in sorted(shelve_db, key=str.lower)]
        finally:
            if shelve_db is not None:
                shelve_db.close()
        return []

    def load(self, title):
        shelve_db = None
        try:
            shelve_db = shelve.open(self.shelve_name)
            if title in shelve_db:
                return DVD(title, *shelve_db[title])
        finally:
            if shelve_db is not None:
                shelve_db.close()
        return None
    def remove(self, title):
        shelve_db = None
        try:
            shelve_db = shelve.open(self.shelve_name)
            del shelve_db[title]
            shelve_db.sync()
        finally:
            if shelve_db is not None:
                shelve_db.close()
```

DB-API 2.0（PEP 249）

- The `sqlite3` module provides a SQL interface compliant with the DB-API 2.0.



```
def connect(name):
    create = not os.path.exists(name)
    conn = sqlite3.connect(name)
    if create:
        cursor = conn.cursor()
        cursor.execute("CREATE TABLE directors ("
            "id INTEGER PRIMARY KEY AUTOINCREMENT UNIQUE NOT NULL, "
            "name TEXT UNIQUE NOT NULL)")
        cursor.execute("CREATE TABLE dvds ("
            "id INTEGER PRIMARY KEY AUTOINCREMENT UNIQUE NOT NULL, "
            "title TEXT NOT NULL, "
            "year INTEGER NOT NULL, "
            "duration INTEGER NOT NULL, "
            "director_id INTEGER NOT NULL, "
            "FOREIGN KEY (director_id) REFERENCES directors)")
        conn.commit()

    return conn
```

```
def add_dvd(conn, title, year, duration, director):
    director_id = get_and_set_director(conn, director)
    cursor = conn.cursor()
    cursor.execute("INSERT INTO dvds "
                   "(title, year, duration, director_id) "
                   "VALUES (?, ?, ?, ?)",
                   (title, year, duration, director_id))
    conn.commit()

def get_and_set_director(conn, director):
    director_id = get_director_id(conn, director)
    if director_id is not None:
        return director_id
    cursor = conn.cursor()
    cursor.execute("INSERT INTO directors (name) VALUES (?)",
                   (director,))
    conn.commit()
    return get_director_id(conn, director)

def get_director_id(conn, director):
    cursor = conn.cursor()
    cursor.execute("SELECT id FROM directors WHERE name=?",
                   (director,))
    fields = cursor.fetchone()
    return fields[0] if fields is not None else None
```

```
def all_dvds(conn):
    cursor = conn.cursor()
    sql = ("SELECT dvds.title, dvds.year, dvds.duration, "
           "directors.name FROM dvds, directors "
           "WHERE dvds.director_id = directors.id"
           " ORDER BY dvds.title")
    cursor.execute(sql)
    return [(str(fields[0]), fields[1], fields[2], str(fields[3]))
            for fields in cursor]

def all_directors(conn):
    cursor = conn.cursor()
    cursor.execute("SELECT name FROM directors ORDER BY name")
    return [str(fields[0]) for fields in cursor]
```

Exercise 8

- There’re three incomplete source files located in lab/exercises/exercise8. Choose what you are interested in and complete it.
- All code you need were listed in the previous slides.

References

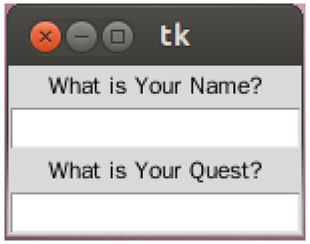
- The Community
  - www.python.org/~guido/
  - http://www.python.org/psf/
  - http://www.python.org/dev/peps/
  - http://www.pycon.org/
  - http://wiki.python.org/moin/LocalUserGroups/
- Documentation
  - http://docs.python.org/2.7/
  - http://docs.python.org/2/library/pydoc.html
- Data Management Functions
  - http://docs.python.org/2.7/library/functions.html
- Persistence
  - http://docs.python.org/2/library/pickle.html
  - http://docs.python.org/2.7/library/dbm.html
  - http://docs.python.org/2/library/shelve.html
  - http://docs.python.org/2.7/library/sqlite3.html

Libraries vs Frameworks

- **What is the difference between a framework and a library?**
- Using libraries, your code is in control: it decides when to ask questions, when to read responses, and when to process those results.

```
name = raw_input('What is your name?')
process_name(name)
quest = raw_input('What is your quest?')
process_quest(quest)
```

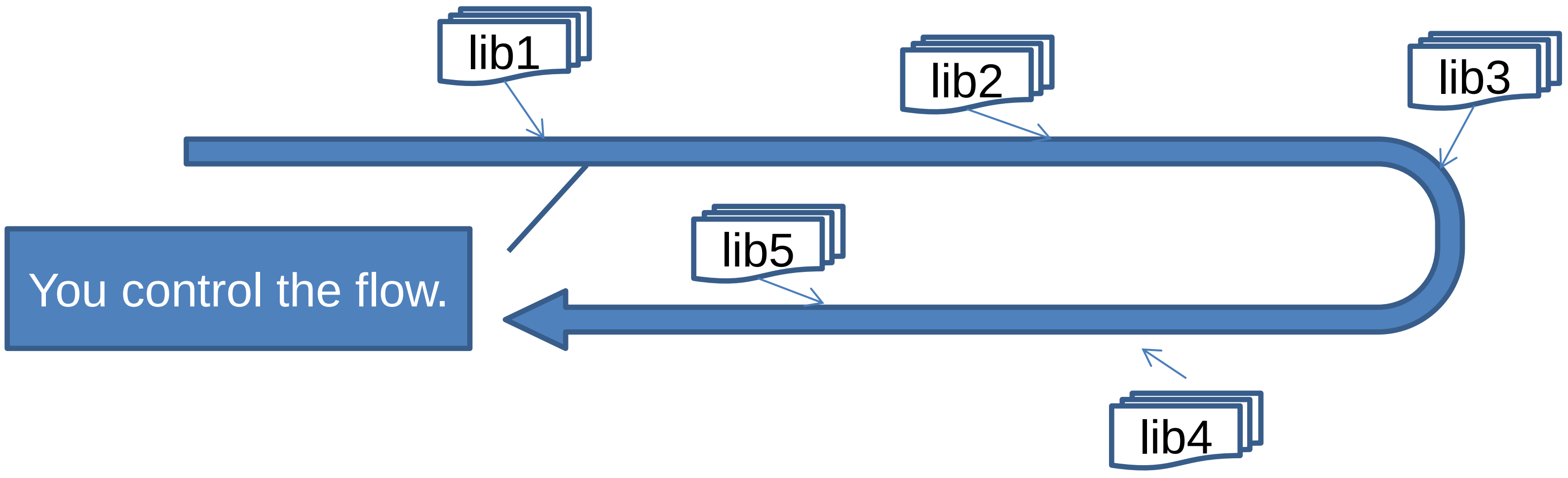
- Using frameworks, it decides when to call your methods, based on the bindings you made when creating the form. **The control is inverted - it calls you rather you calling the framework**.



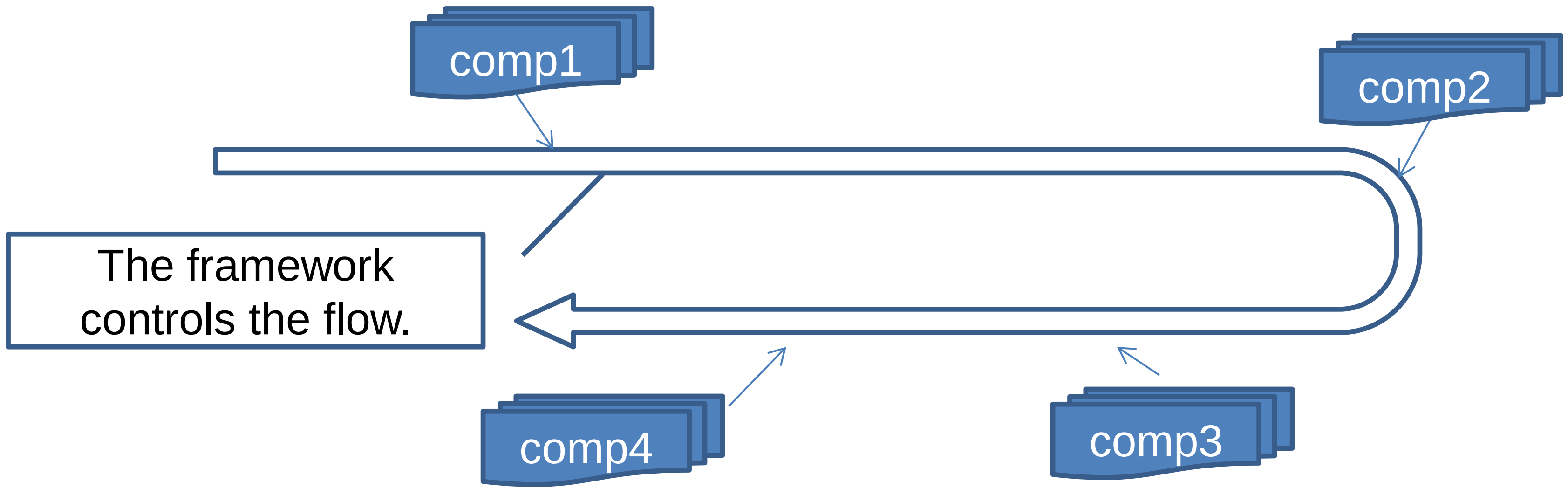
```
import Tkinter

top = Tkinter.Tk()

Tkinter.Label(top, text='What is Your Name?').pack()
name_var = Tkinter.StringVar()
name_entry = Tkinter.Entry(top, textvariable=name_var)
name_entry.pack()
name_entry.bind('<FocusOut>', lambda event: process_name(name_var))

Tkinter.Label(top, text='What is Your Quest?').pack()
quest_var = Tkinter.StringVar()
quest_entry = Tkinter.Entry(top, textvariable=quest_var)
quest_entry.pack()
quest_entry.bind('<FocusOut>', lambda event:
process_name(quest_var))

Tkinter.mainloop()
```

Inversion of Control

- Using libraries

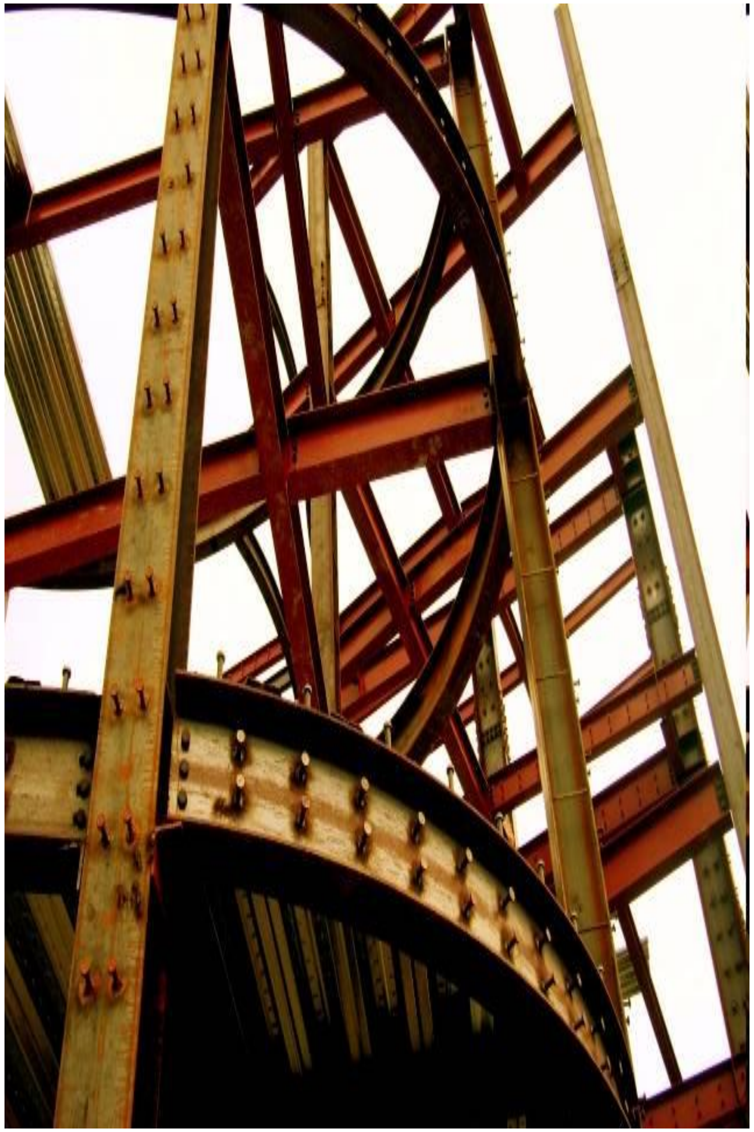


- Using frameworks



Do We Need a Framework?

- **Libraries bring developers freedom.**
- **Frameworks bring developers constraints.**
  - Do we need a framework?
  - Do we want to follow the flow?
  - Do we make decisions according to technical reasons, or business reasons?
- A right framework brings you a heaven; the wrong one brings you a hell.



Getting Started with Django

- Django (www.djangoproject.com) is a high-level Python Web framework that encourages rapid development and clean, pragmatic design.
  - Object-relational mapper
  - Automatic admin interface
  - Elegant URL design
  - Template system
  - Cache system
  - Internationalization

Design Your Models

```
class Reporter(models.Model):
    full_name = models.CharField(max_length=70)

    def __unicode__(self):
        return self.full_name

class Article(models.Model):
    pub_date = models.DateField()
    headline = models.CharField(max_length=200)
    content = models.TextField()
    reporter = models.ForeignKey(Reporter)

    def __unicode__(self):
        return self.headline
```

```
# No reporters are in the system yet.
>>> Reporter.objects.all()
[]

# Create a new Reporter.
>>> r = Reporter(full_name='John Smith')

# Save the object into the database. You have to call save() explicitly.
>>> r.save()

# Now it has an ID.
>>> r.id
1
```

Design Your URLs

```
from django.conf.urls import patterns

urlpatterns = patterns('',
    (r'^articles/(\d{4})/$', 'news.views.year_archive'),
    (r'^articles/(\d{4})/(\d{2})/$', 'news.views.month_archive'),
    (r'^articles/(\d{4})/(\d{2})/(\d+)/$', 'news.views.article_detail'),
)
```

Write Your Views and Templates

```
def year_archive(request, year):
    a_list = Article.objects.filter(pub_date__year=year)
    return render_to_response('news/year_archive.html', {'year': year, 'article_list': a_list})
```

```
<html>
<head>
    <title>{% block title %}{% endblock %}</title>
</head>
<body>
    <img src="sitelogo.png" alt="Logo" />
    {% block content %}{% endblock %}
</body>
</html>
```

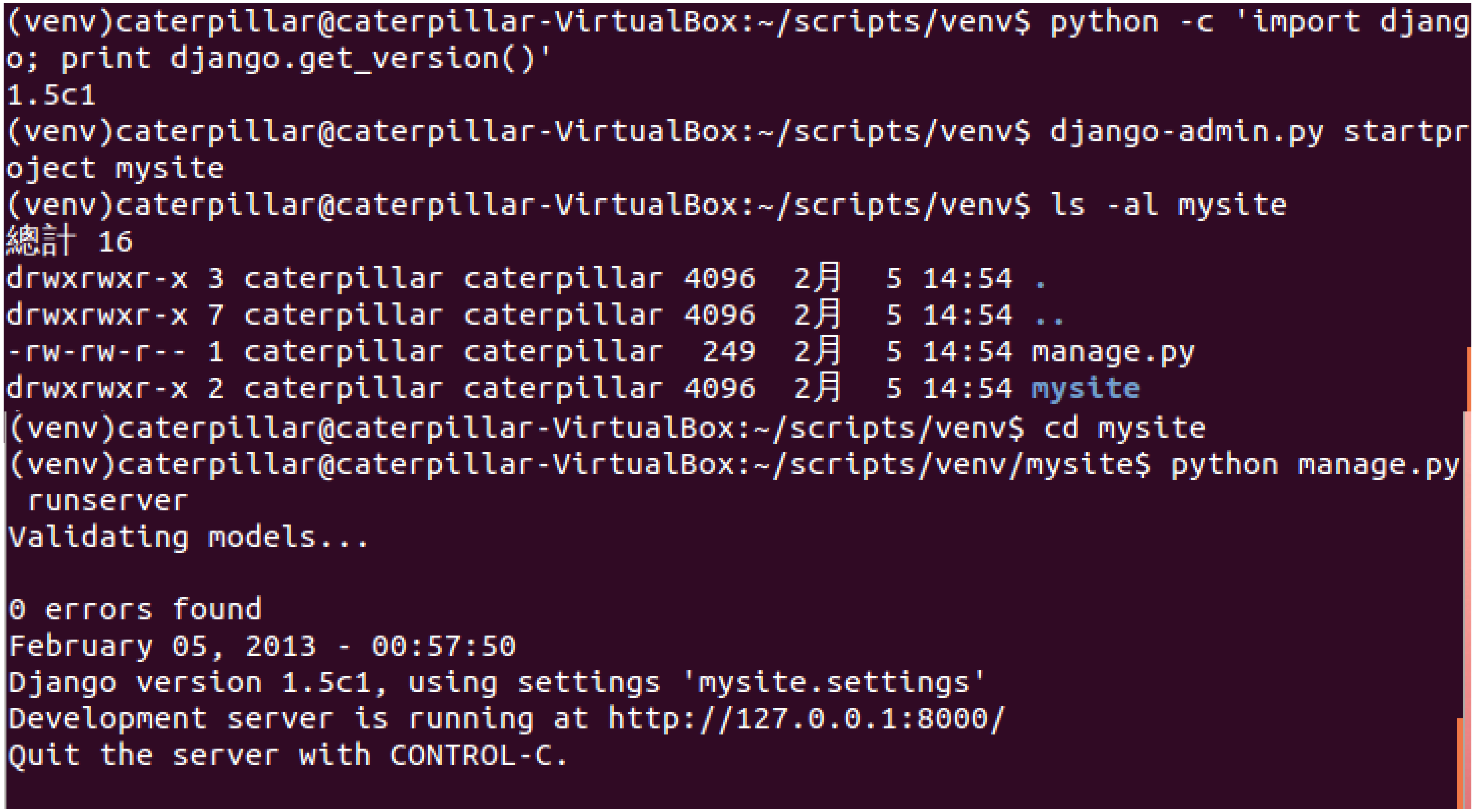
```
{% extends "base.html" %}

{% block title %}Articles for {{ year }}{% endblock %}

{% block content %}
<h1>Articles for {{ year }}</h1>

{% for article in article_list %}
    <p>{{ article.headline }}</p>
    <p>By {{ article.reporter.full_name }}</p>
    <p>Published {{ article.pub_date|date:"F j, Y" }}</p>
{% endfor %}
{% endblock %}
```

Creating a Project (Exercise 9)

- We’d like to install an offical realse of **Django 1.5 rc1** with **pip** under a virtual Python environment provided by **virtualenv**. And Then, create our first django project.

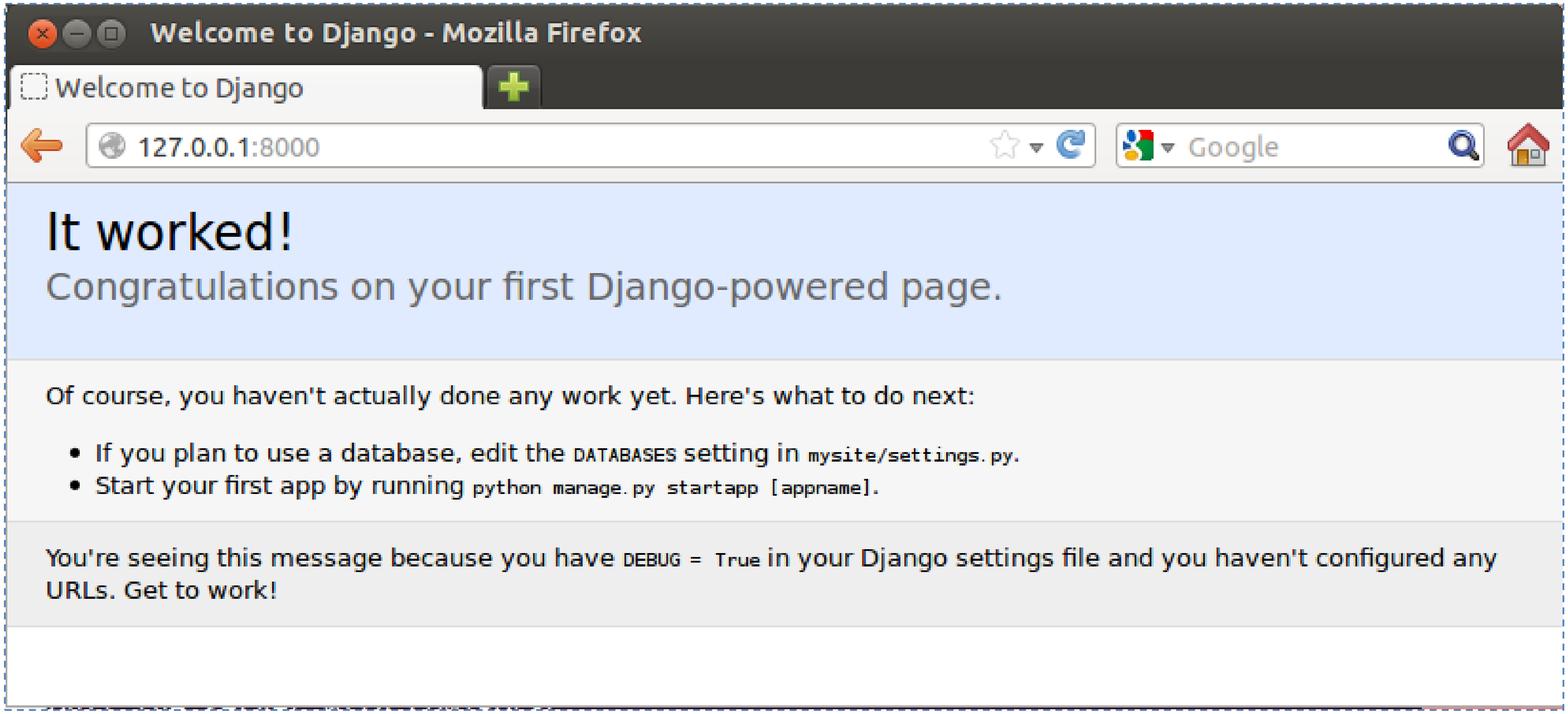
```
~/scripts$ virtualenv --distribute venv
~/scripts$ cd venv
~/scripts/venv$ source bin/activate
~/scripts/venv$ pip install https://www.djangoproject.com/download/1.5c1/tarball/
~/scripts/venv$ python -c 'import django; print django.get_version()‘
~/scripts/venv$ django-admin.py startproject mysite
~/scripts/venv$ ls -al mysite
~/scripts/venv$ cd mysite
~/scripts/venv$ python manage.py runserver
```

What You Should See



```
(venv)caterpillar@caterpillar-VirtualBox:~/scripts/venv$ python -c 'import djang
o; print django.get_version()'
1.5c1
(venv)caterpillar@caterpillar-VirtualBox:~/scripts/venv$ django-admin.py startpr
oject mysite
(venv)caterpillar@caterpillar-VirtualBox:~/scripts/venv$ ls -al mysite
總計 16
drwxrwxr-x 3 caterpillar caterpillar 4096  2月   5 14:54 .
drwxrwxr-x 7 caterpillar caterpillar 4096  2月   5 14:54 ..
-rw-rw-r-- 1 caterpillar caterpillar  249  2月   5 14:54 manage.py
drwxrwxr-x 2 caterpillar caterpillar 4096  2月   5 14:54 mysite
(venv)caterpillar@caterpillar-VirtualBox:~/scripts/venv$ cd mysite
(venv)caterpillar@caterpillar-VirtualBox:~/scripts/venv/mysite$ python manage.py
 runserver
Validating models...

0 errors found
February 05, 2013 - 00:57:50
Django version 1.5c1, using settings 'mysite.settings'
Development server is running at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

What You Should See



It worked!

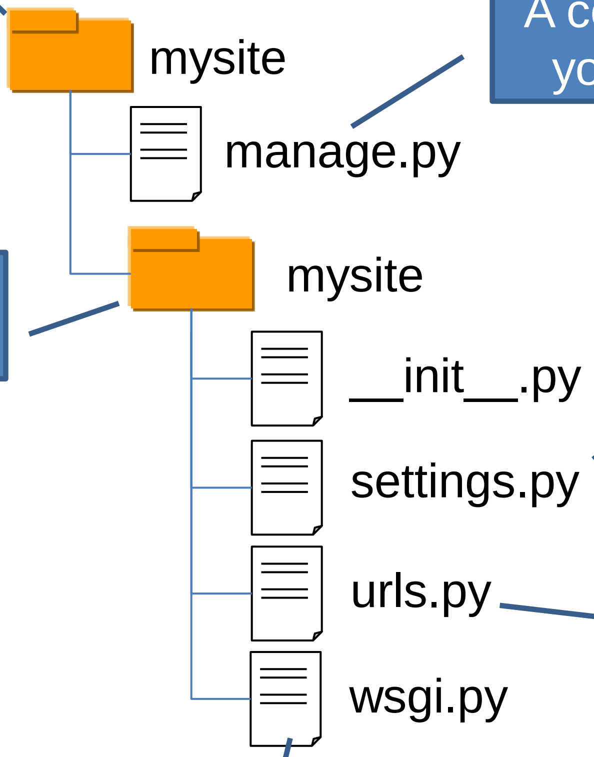
Congratulations on your first Django-powered page.

Of course, you haven't actually done any work yet. Here's what to do next:

- If you plan to use a database, edit the `DATABASES` setting in `mysite/settings.py`.
- Start your first app by running `python manage.py startapp [appname]`.

You're seeing this message because you have `DEBUG = True` in your Django settings file and you haven't configured any URLs. Get to work!

What `startproject` Created



You can rename it to anything you like.

A command-line utility that lets you interact with this project

The actual Python package for your project.

Settings/configuration for this project.

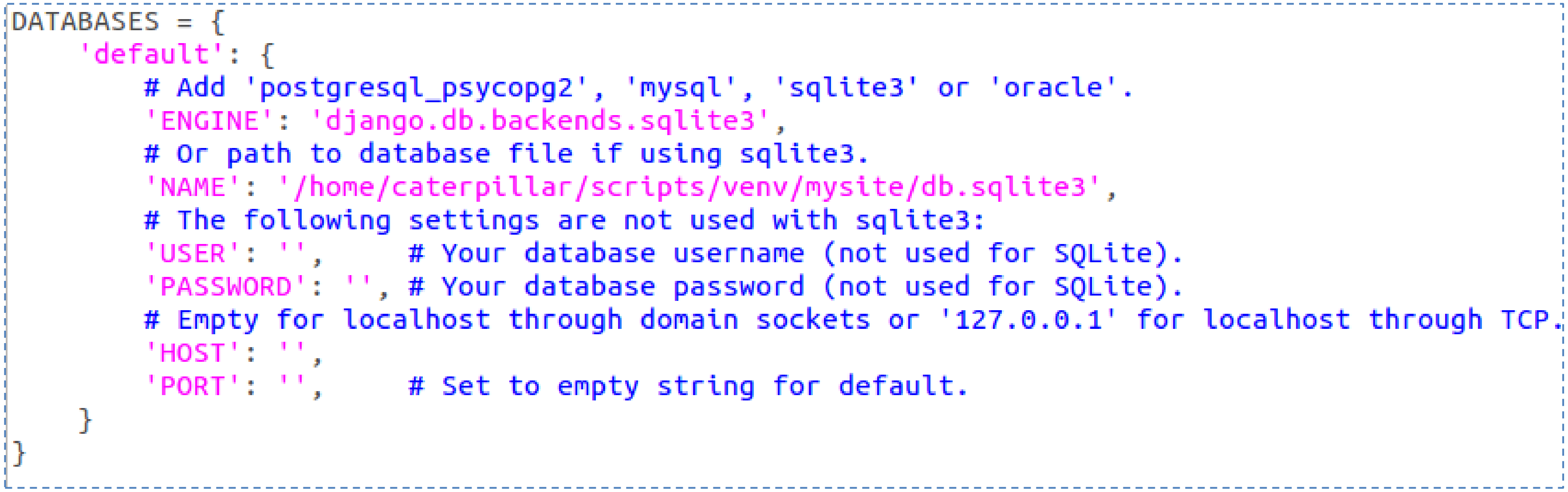
The URL declarations for this project

Django's primary deployment platform is WSGI.

An entry-point for WSGI-compatible webservers to serve your project.

Creating a Database and an App (Exercise 10)

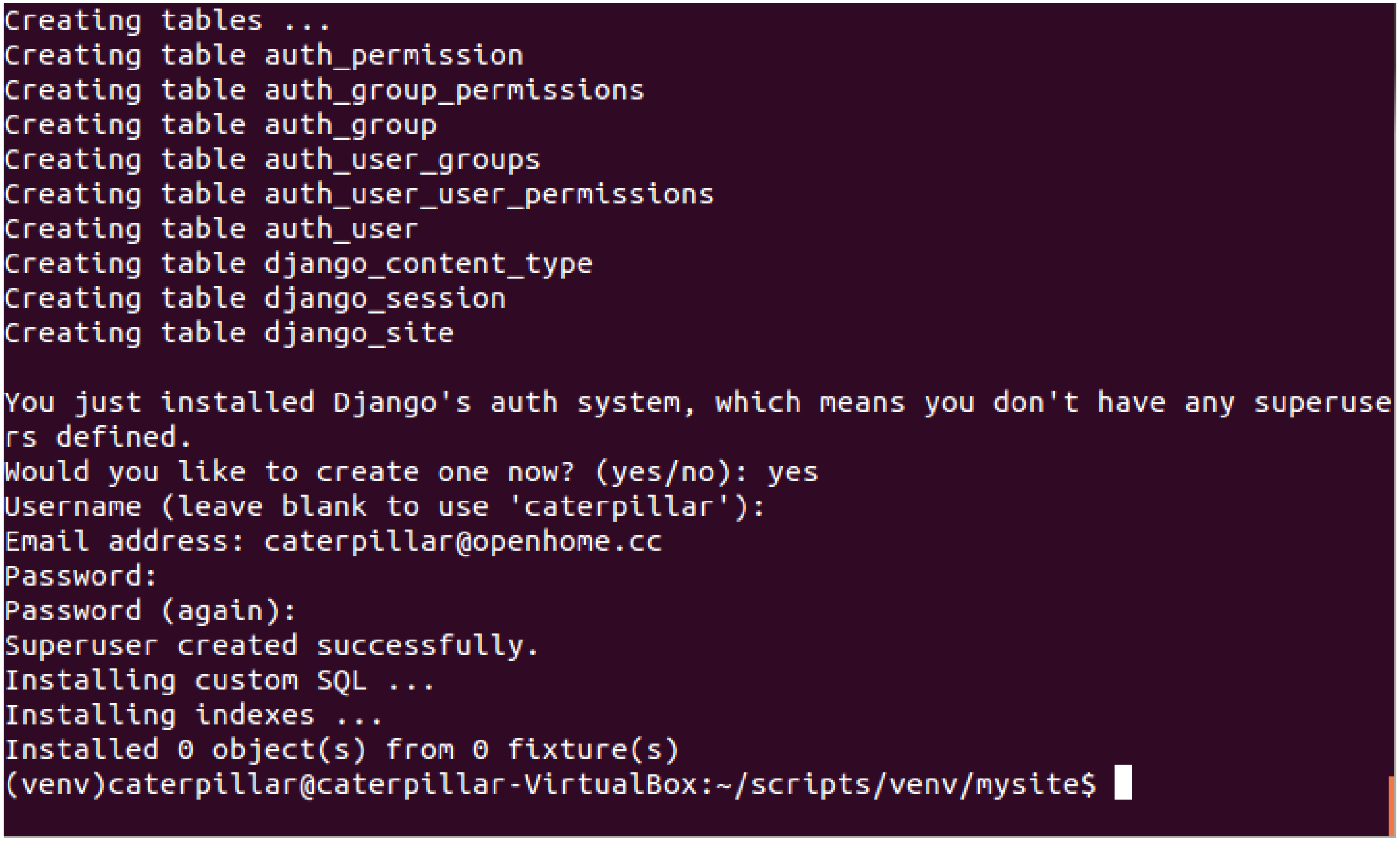
- Edit **mysite/settings.py**. Change the following keys in the `DATABASES 'default'` item to match your database connection settings.



```
DATABASES = {
    'default': {
        # Add 'postgresql_psycopg2', 'mysql', 'sqlite3' or 'oracle'.
        'ENGINE': 'django.db.backends.sqlite3',
        # Or path to database file if using sqlite3.
        'NAME': '/home/caterpillar/scripts/venv/mysite/db.sqlite3',
        # The following settings are not used with sqlite3:
        'USER': '',      # Your database username (not used for SQLite).
        'PASSWORD': '', # Your database password (not used for SQLite).
        # Empty for localhost through domain sockets or '127.0.0.1' for localhost through TCP.
        'HOST': '',
        'PORT': '',      # Set to empty string for default.
    }
}
```

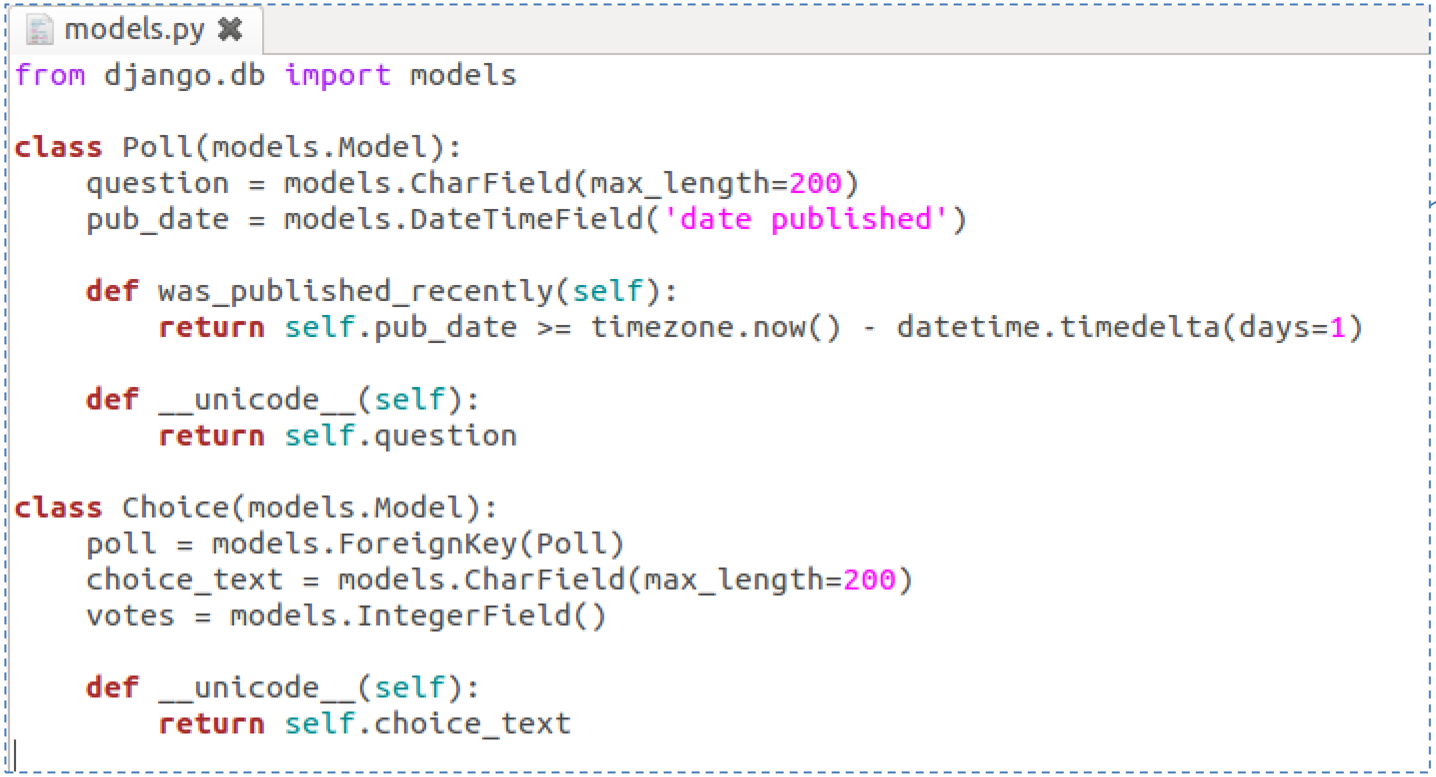
- `python manage.py syncdb`

What You Should See



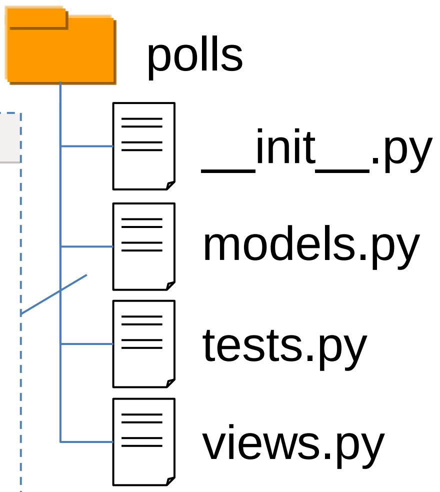
```
Creating tables ...
Creating table auth_permission
Creating table auth_group_permissions
Creating table auth_group
Creating table auth_user_groups
Creating table auth_user_user_permissions
Creating table auth_user
Creating table django_content_type
Creating table django_session
Creating table django_site

You just installed Django's auth system, which means you don't have any superuse
rs defined.
Would you like to create one now? (yes/no): yes
Username (leave blank to use 'caterpillar'):
Email address: caterpillar@openhome.cc
Password:
Password (again):
Superuser created successfully.
Installing custom SQL ...
Installing indexes ...
Installed 0 object(s) from 0 fixture(s)
(venv)caterpillar@caterpillar-VirtualBox:~/scripts/venv/mysite$
```

- Type the following command to create a simple poll app.
  - `python manage.py startapp polls`
- Edit the **polls/models.py** so it looks like this:

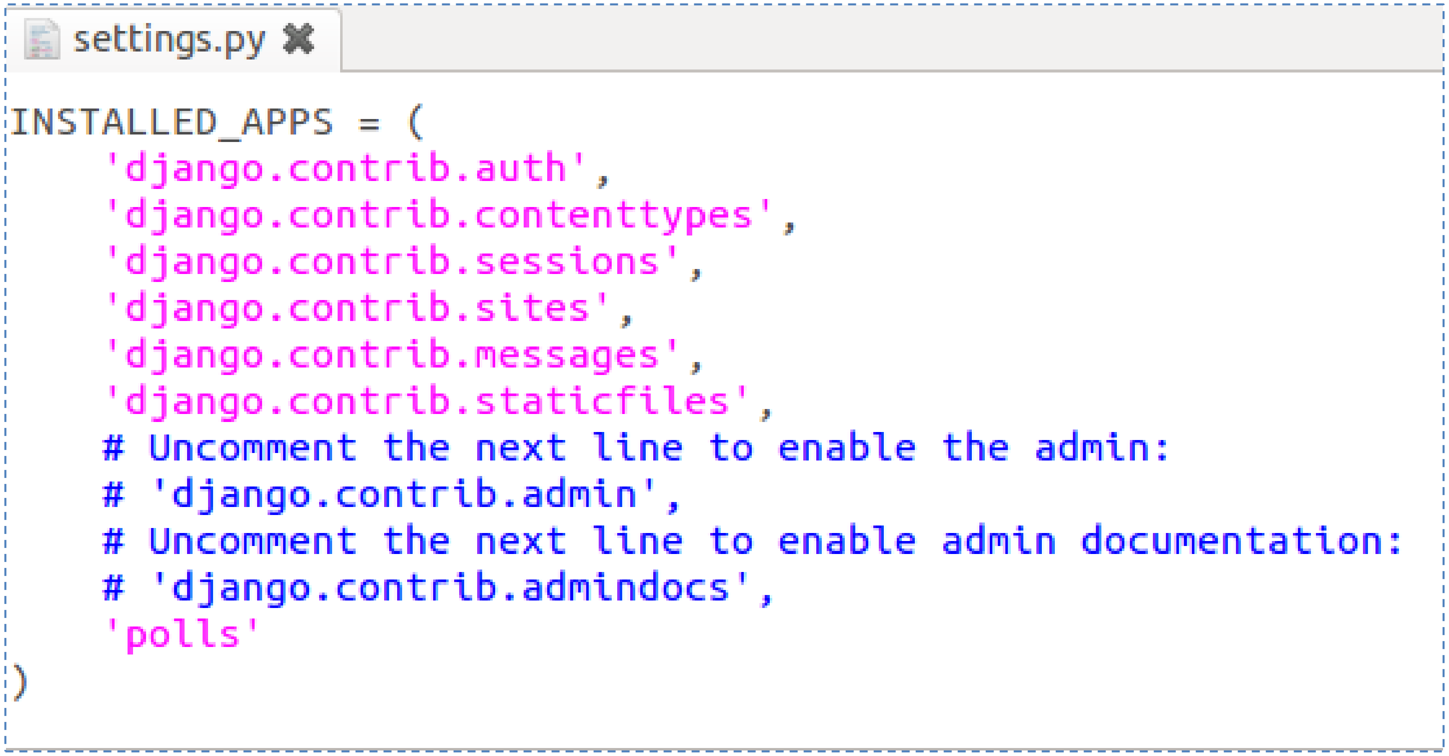


models.py

```
from django.db import models

class Poll(models.Model):
    question = models.CharField(max_length=200)
    pub_date = models.DateTimeField('date published')

    def was_published_recently(self):
        return self.pub_date >= timezone.now() - datetime.timedelta(days=1)

    def __unicode__(self):
        return self.question

class Choice(models.Model):
    poll = models.ForeignKey(Poll)
    choice_text = models.CharField(max_length=200)
    votes = models.IntegerField()

    def __unicode__(self):
        return self.choice_text
```



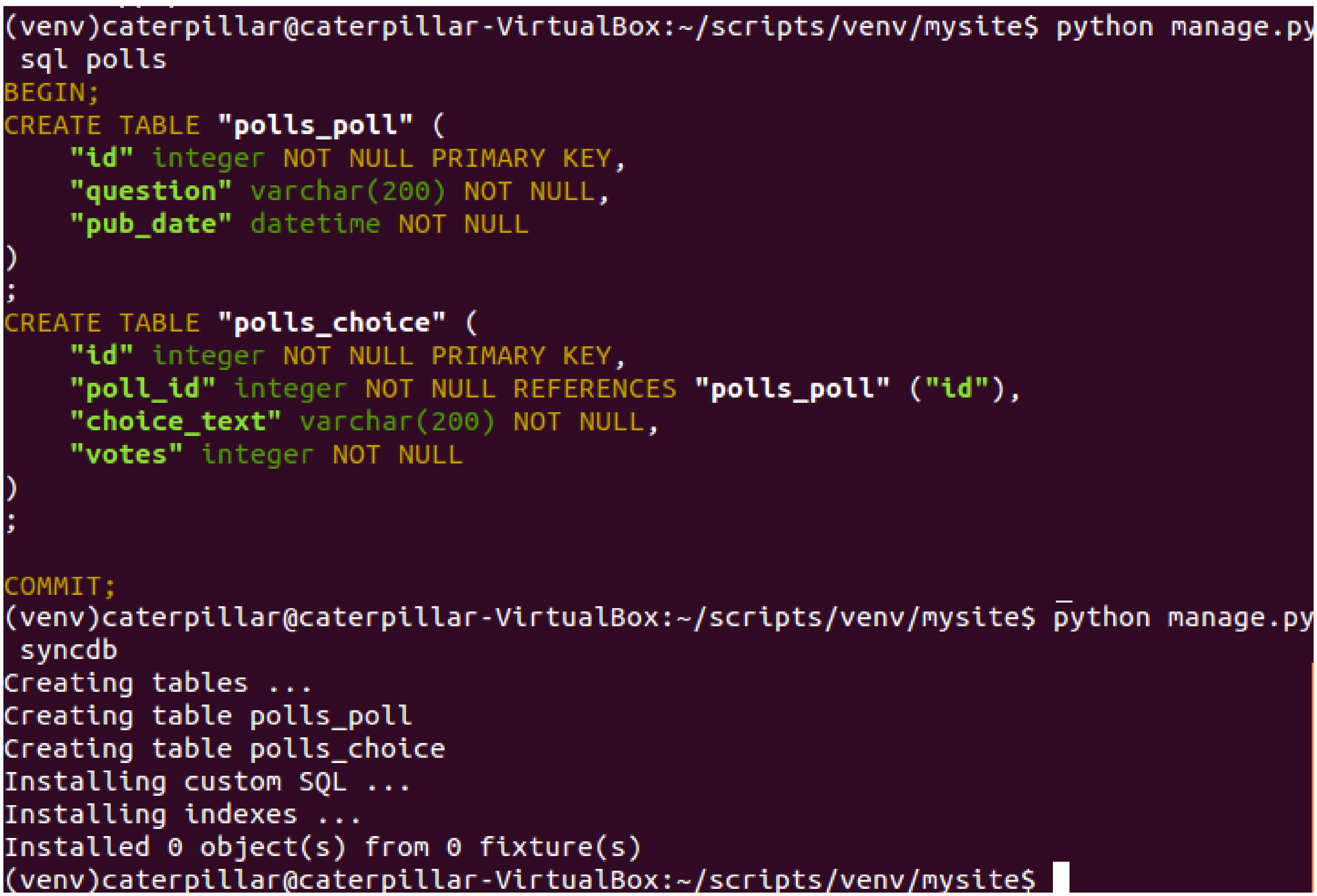
- Edit the **settings.py** again, and change the `INSTALLED_APPS` setting to include the string `'polls'`.



```
INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    # Uncomment the next line to enable the admin:
    # 'django.contrib.admin',
    # Uncomment the next line to enable admin documentation:
    # 'django.contrib.admindocs',
    'polls'
)
```

- Type the following command to create tables for the polls app.
  - `python manage.py sql polls`
  - `python manage.py syncdb`

What You Should See

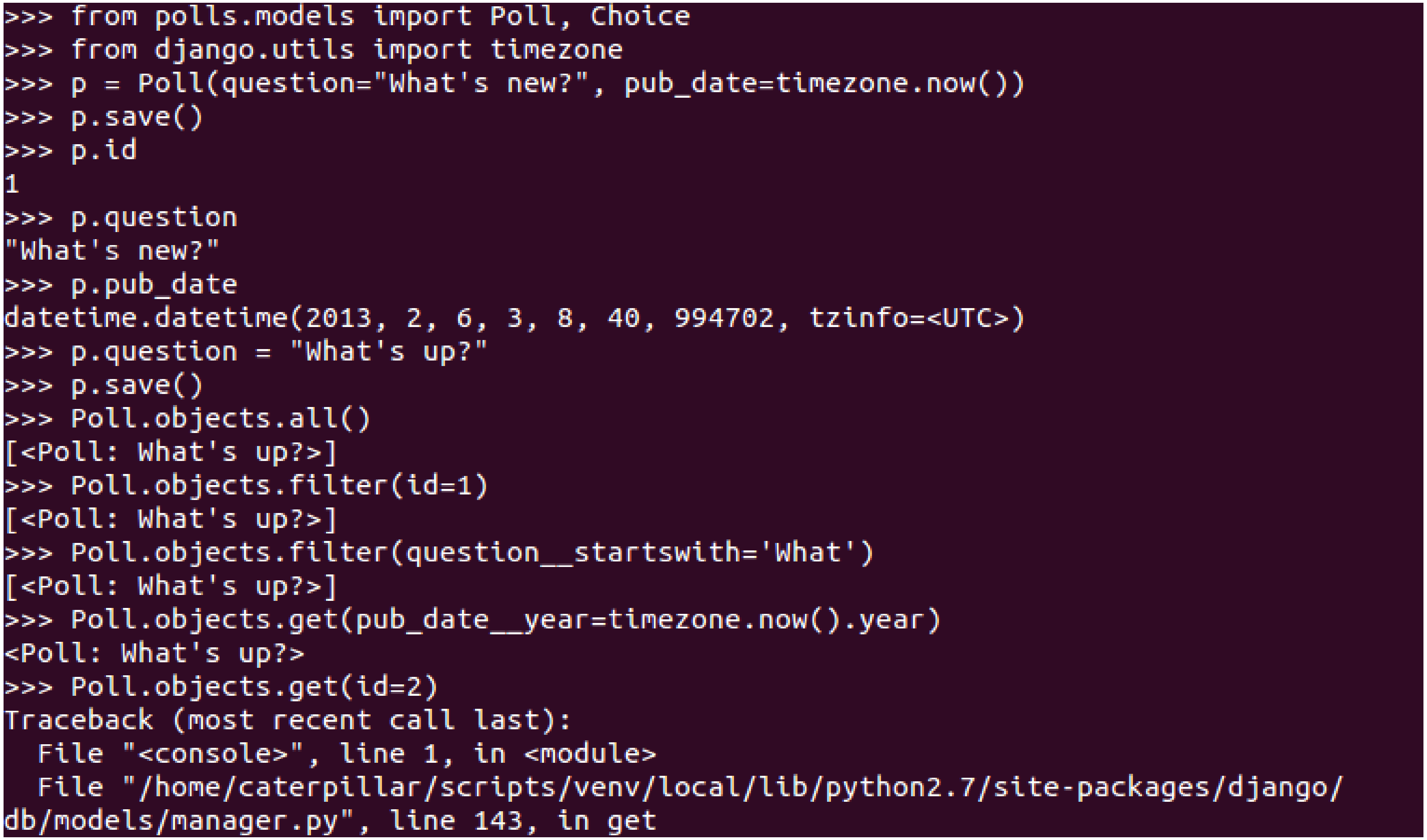


```
(venv)caterpillar@caterpillar-VirtualBox:~/scripts/venv/mysite$ python manage.py
 sql polls
BEGIN;
CREATE TABLE "polls_poll" (
    "id" integer NOT NULL PRIMARY KEY,
    "question" varchar(200) NOT NULL,
    "pub_date" datetime NOT NULL
)
;
CREATE TABLE "polls_choice" (
    "id" integer NOT NULL PRIMARY KEY,
    "poll_id" integer NOT NULL REFERENCES "polls_poll" ("id"),
    "choice_text" varchar(200) NOT NULL,
    "votes" integer NOT NULL
)
;

COMMIT;
(venv)caterpillar@caterpillar-VirtualBox:~/scripts/venv/mysite$ python manage.py
 syncdb
Creating tables ...
Creating table polls_poll
Creating table polls_choice
Installing custom SQL ...
Installing indexes ...
Installed 0 object(s) from 0 fixture(s)
(venv)caterpillar@caterpillar-VirtualBox:~/scripts/venv/mysite$
```

Playing API with the Python shell

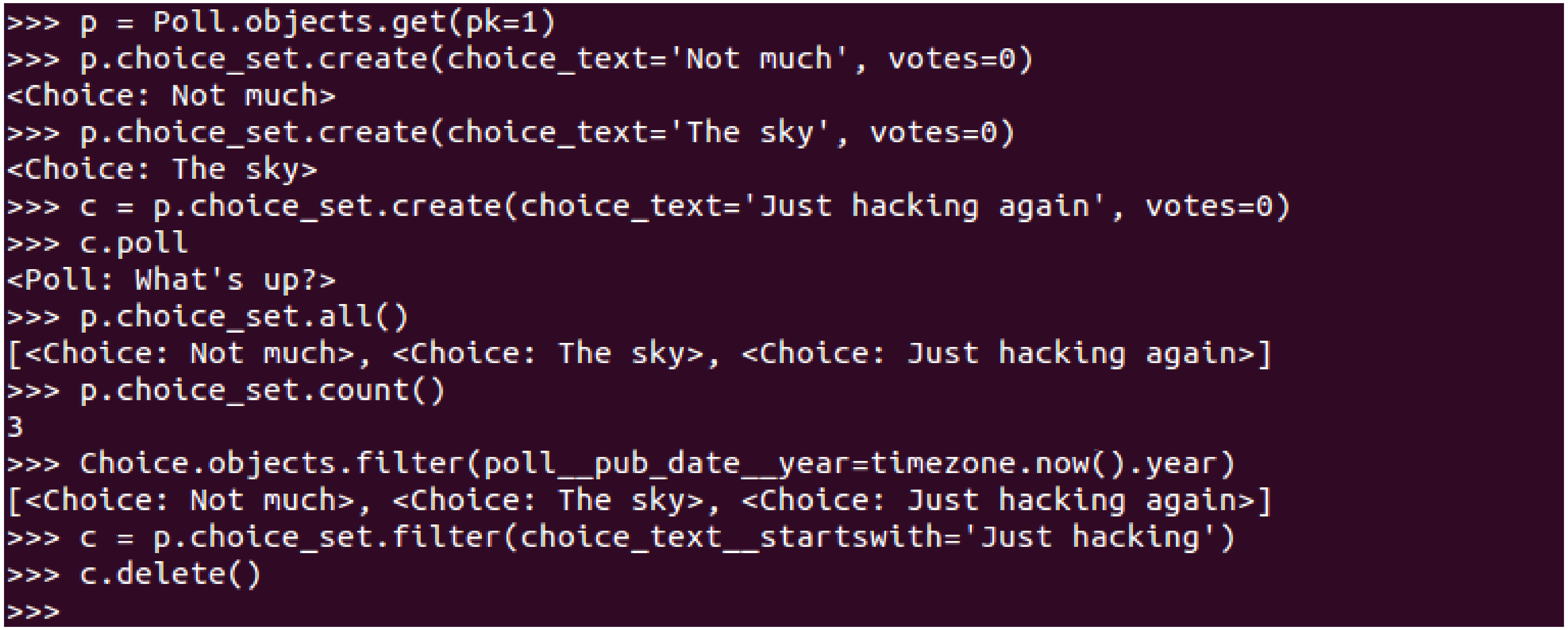
- Type the following command to set the `DJANGO_SETTINGS_MODULE` environment variable, which gives Django the Python import path to your settings.py file.
  - `python manage.py shell`

Basic ORM



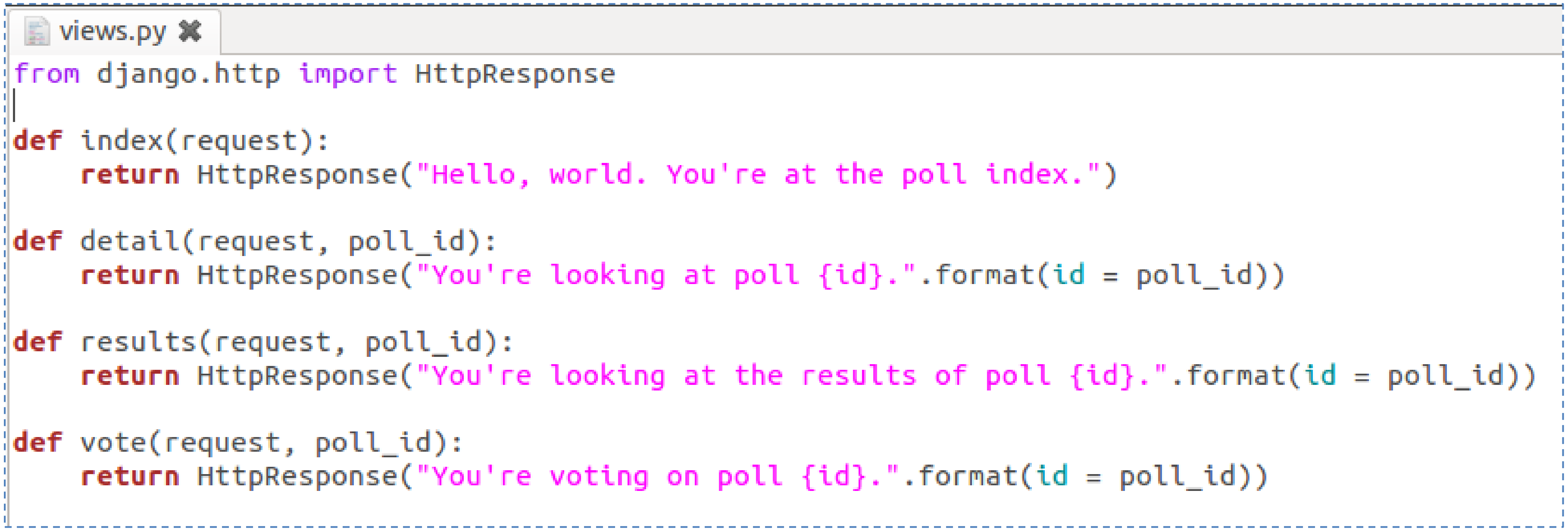
```
>>> from polls.models import Poll, Choice
>>> from django.utils import timezone
>>> p = Poll(question="What's new?", pub_date=timezone.now())
>>> p.save()
>>> p.id
1
>>> p.question
"What's new?"
>>> p.pub_date
datetime.datetime(2013, 2, 6, 3, 8, 40, 994702, tzinfo=<UTC>)
>>> p.question = "What's up?"
>>> p.save()
>>> Poll.objects.all()
[<Poll: What's up?>]
>>> Poll.objects.filter(id=1)
[<Poll: What's up?>]
>>> Poll.objects.filter(question__startswith='What')
[<Poll: What's up?>]
>>> Poll.objects.get(pub_date__year=timezone.now().year)
<Poll: What's up?>
>>> Poll.objects.get(id=2)
Traceback (most recent call last):
  File "<console>", line 1, in <module>
  File "/home/caterpillar/scripts/venv/local/lib/python2.7/site-packages/django/
db/models/manager.py", line 143, in get
```

One-to-One Relationship


```
>>> p = Poll.objects.get(pk=1)
>>> p.choice_set.create(choice_text='Not much', votes=0)
<Choice: Not much>
>>> p.choice_set.create(choice_text='The sky', votes=0)
<Choice: The sky>
>>> c = p.choice_set.create(choice_text='Just hacking again', votes=0)
>>> c.poll
<Poll: What's up?>
>>> p.choice_set.all()
[<Choice: Not much>, <Choice: The sky>, <Choice: Just hacking again>]
>>> p.choice_set.count()
3
>>> Choice.objects.filter(poll__pub_date__year=timezone.now().year)
[<Choice: Not much>, <Choice: The sky>, <Choice: Just hacking again>]
>>> c = p.choice_set.filter(choice_text__startswith='Just hacking')
>>> c.delete()
>>>
```

Writing Your First View (Exercise 11)

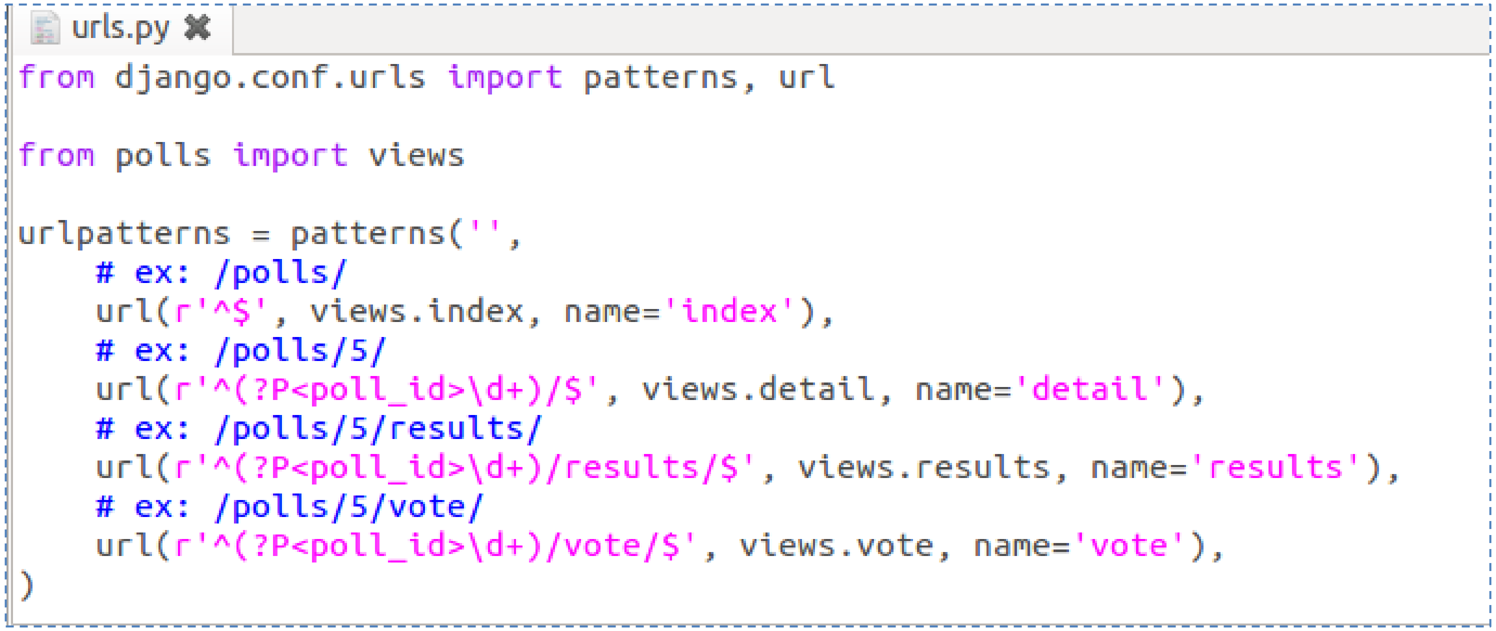
- Let’s write your first view. Open the file **polls/views.py** and put the following Python code in it:



views.py

```
from django.http import HttpResponse

def index(request):
    return HttpResponse("Hello, world. You're at the poll index.")

def detail(request, poll_id):
    return HttpResponse("You're looking at poll {id}.".format(id = poll_id))

def results(request, poll_id):
    return HttpResponse("You're looking at the results of poll {id}.".format(id = poll_id))

def vote(request, poll_id):
    return HttpResponse("You're voting on poll {id}.".format(id = poll_id))
```

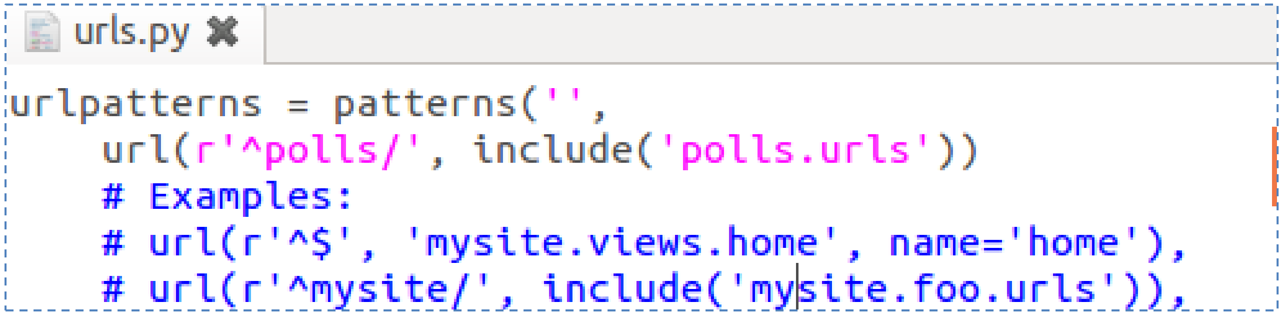
- Create a file called **urls.py** in the **polls** directory. Include the following code:



urls.py ✖

```
from django.conf.urls import patterns, url

from polls import views

urlpatterns = patterns('',
    # ex: /polls/
    url(r'^$', views.index, name='index'),
    # ex: /polls/5/
    url(r'^(?P<poll_id>\d+)/$', views.detail, name='detail'),
    # ex: /polls/5/results/
    url(r'^(?P<poll_id>\d+)/results/$', views.results, name='results'),
    # ex: /polls/5/vote/
    url(r'^(?P<poll_id>\d+)/vote/$', views.vote, name='vote'),
)
```

- Open **urls.py** in the **mysite** directory. Include the following code:

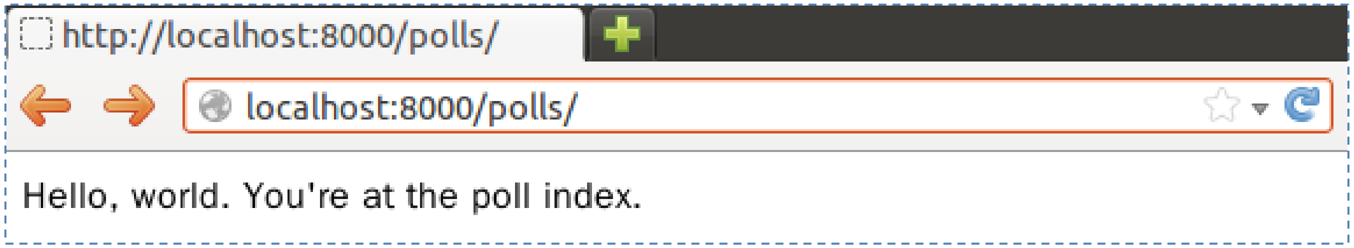


urls.py ✖

```
urlpatterns = patterns('',
    url(r'^polls/', include('polls.urls'))
    # Examples:
    # url(r'^$', 'mysite.views.home', name='home'),
    # url(r'^mysite/', include('mysite.foo.urls')),
```

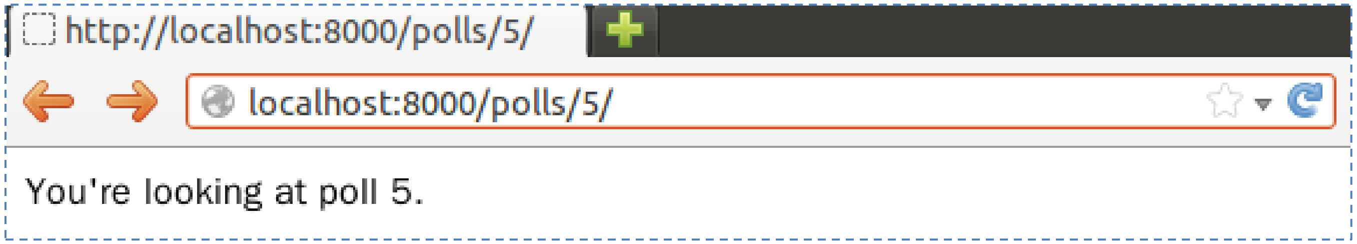
- Type the following command to start the Django development server.
  - `python manage.py runserver`
- Visit the following urls with your browser.
  - http://localhost:8000/polls/
  - http://localhost:8000/polls/5/
  - http://localhost:8000/polls/5/results/
  - http://localhost:8000/polls/5/vote/

What You Should See


http://localhost:8000/polls/

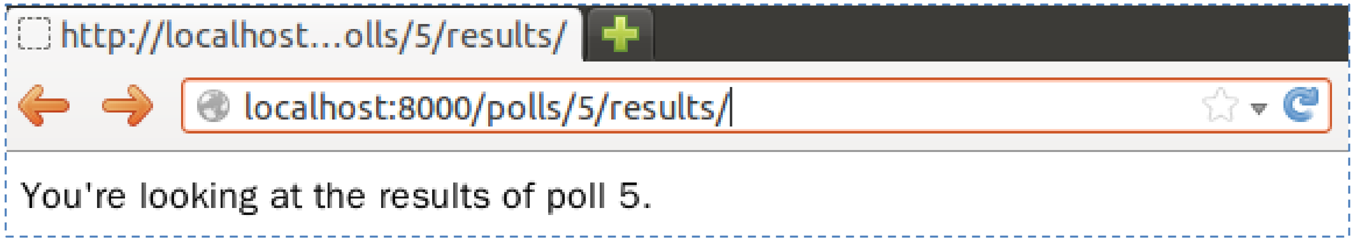
localhost:8000/polls/

Hello, world. You're at the poll index.


http://localhost:8000/polls/5/

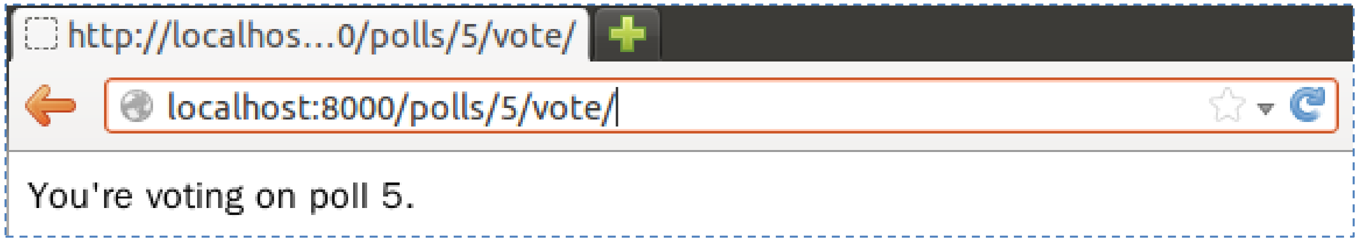
localhost:8000/polls/5/

You're looking at poll 5.


http://localhost...olls/5/results/

localhost:8000/polls/5/results/

You're looking at the results of poll 5.


http://localhos...0/polls/5/vote/

localhost:8000/polls/5/vote/

You're voting on poll 5.

Controllers or Views?

- We are using Django **MVC** framework. Are functions `index`, `details`, `results` and `vote` belong to controllers or views?
  - Well, the standard names are debatable.
  - In Django's case, a "view" is the Python callback function for a particular URL.
  - Where does the "controller" fit in, then? In Django's case, it's probably the framework itself.
  - As you'll see soon, you might say that Django is a **MTV** framework – that is, "**Model**", "**Template**", and "**View**".

- (Is there `before_filter` in Django as in Rails?
  - No. `before_`, `around_` and `after_ filter` concepts aren't present in Django.
  - It’s not hard to hard-code what you need. Or, you can use a generic decorator, such as those provided by the Django authentication system.)

URLconf

- Determining which view is called is done by Python modules informally titled ‘**URLconfs**’.
  - These modules are pure Python code and are a simple mapping between URL patterns to Python callback functions (your views).
- The `url()` function needs two required arguments and one suggested argument.
  - `regex`: URL patterns are simple regular expressions.
  - `view`: When Django finds a regular expression match, Django calls the specified view function, with an `HttpRequest` object as the first argument and any “captured” values from the regular expression as other arguments.
  - `name`: Naming your URL lets you refer to it unambiguously from elsewhere in Django especially templates.

Simple URL Patterns

- For `urlpatterns` in **mysite/urls.py**.

Any request starting with “/polls/

```
url(r'^polls/', include('polls.urls'))
```

Drop “/polls/” and use the remaining to match patterns defined in the `polls.urls` module.

- For `urlpatterns` in **polls/urls.py**.

```
url(r'^$', views.index)
```

An empty string

Call the `views.index` function

```
url(r'^(?P<poll_id>\d+)/$', views.detail)
```

The remaining represents an number, capture it as `poll_id`

Call the `views.details` function. The second argument is the captured `poll_id`.

```
url(r'^(?P<poll_id>\d+)/results/$', views.results)
```

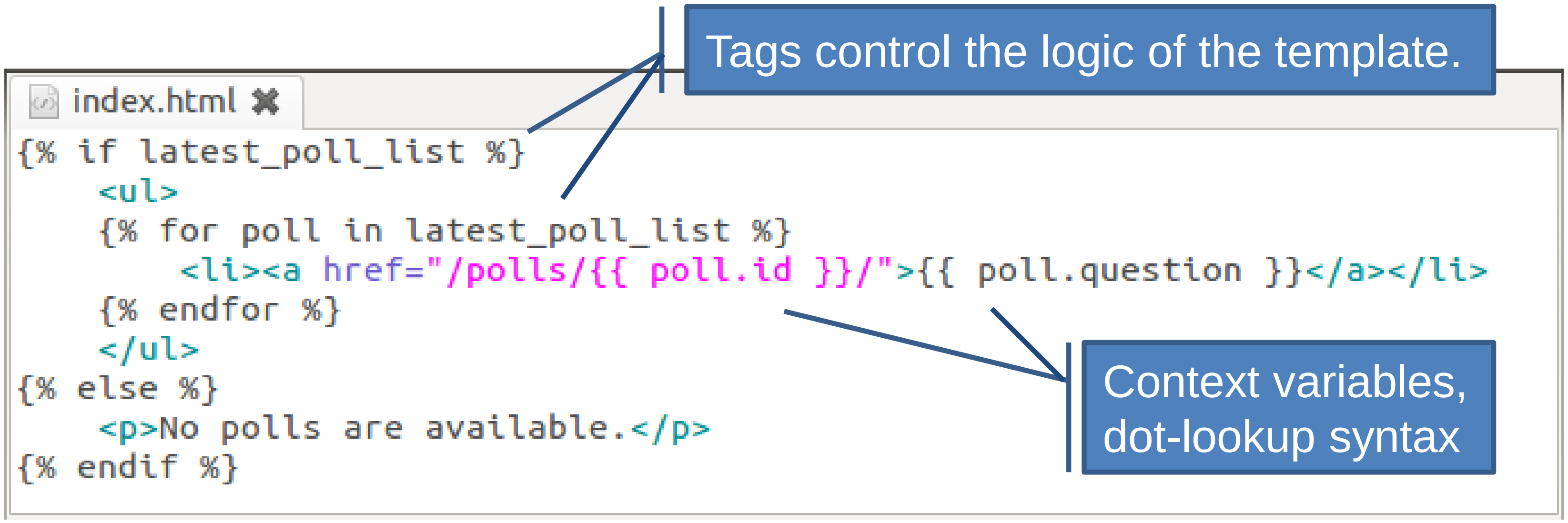
Starting with an number and ends with “/results/”

References

- Libraries vs Frameworks
  - http://martinfowler.com/bliki/InversionOfControl.html
- Getting Started with Django
  - https://docs.djangoproject.com/en/1.5/intro/overview/
  - https://docs.djangoproject.com/en/1.5/
  - https://docs.djangoproject.com/en/1.5/intro/install/
  - http://stackoverflow.com/questions/12339608/installing-django-1-5development-version-in-virtualenv
  - https://docs.djangoproject.com/en/1.5/intro/tutorial01/
- Writing Your First View
  - https://docs.djangoproject.com/en/1.5/intro/tutorial03/
  - https://docs.djangoproject.com/en/dev/faq/general/#django-appears-to-be-a-mvc-framework-but-you-call-the-controller-the-view-and-the-view-the-template-how-come-you-don-t-use-the-standard-names
  - https://docs.djangoproject.com/en/1.5/topics/auth/default/

Using the Template System

- Edit the Python code to change the way the page looks? We don’t want to back to the spaghetti world.
- Let’s use Django’s template system to separate the design from Python.



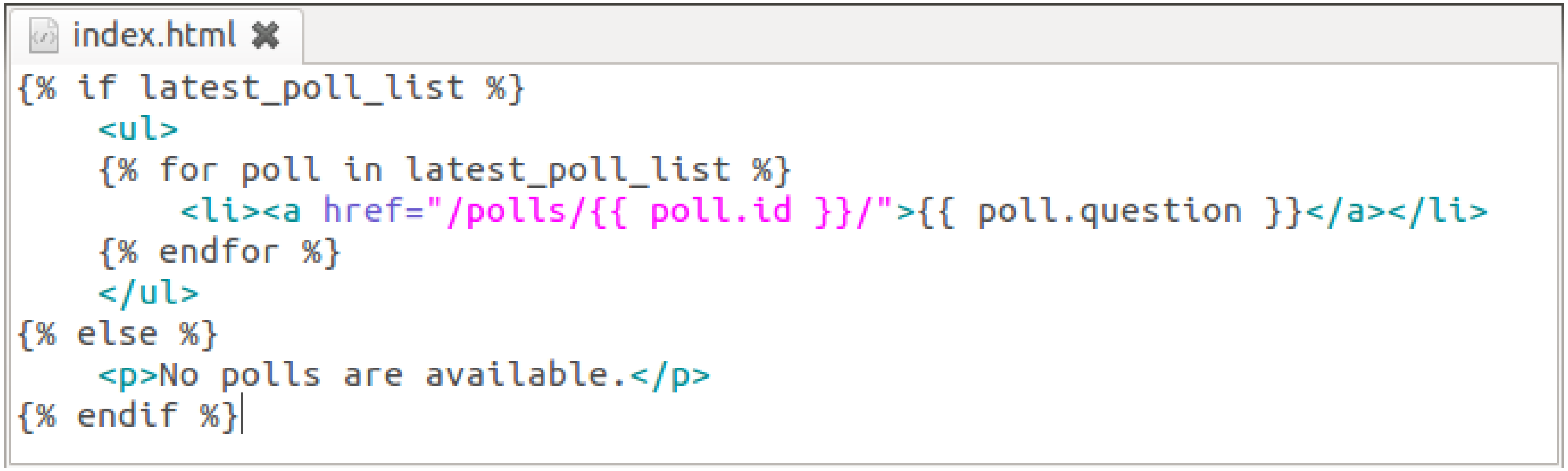
index.html

```
{% if latest_poll_list %}
    <ul>
    {% for poll in latest_poll_list %}
        <li><a href="/polls/{{ poll.id }}/">{{ poll.question }}</a></li>
    {% endfor %}
    </ul>
{% else %}
    <p>No polls are available.</p>
{% endif %}
```

Tags control the logic of the template.
Context variables, dot-lookup syntax

Writing Templates (Exercise 12)

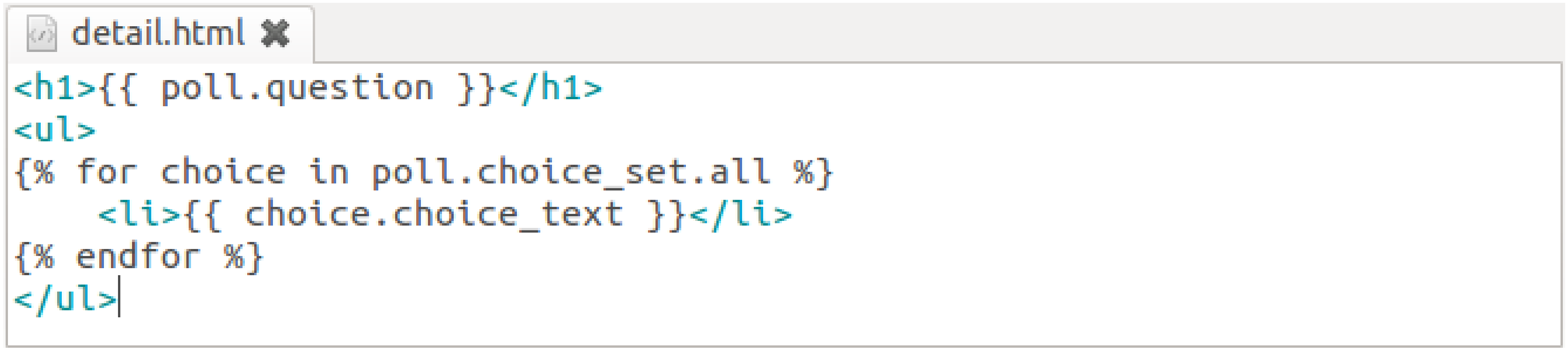
- Create a directory called **templates** in your **polls** directory. Django will look for templates in there.
- Create another directory called **polls**, and within that Create a file called **index.html**.
  - In other words, your template should be at **polls/templates/polls/index.html**.
- Put the following code in that template:



index.html

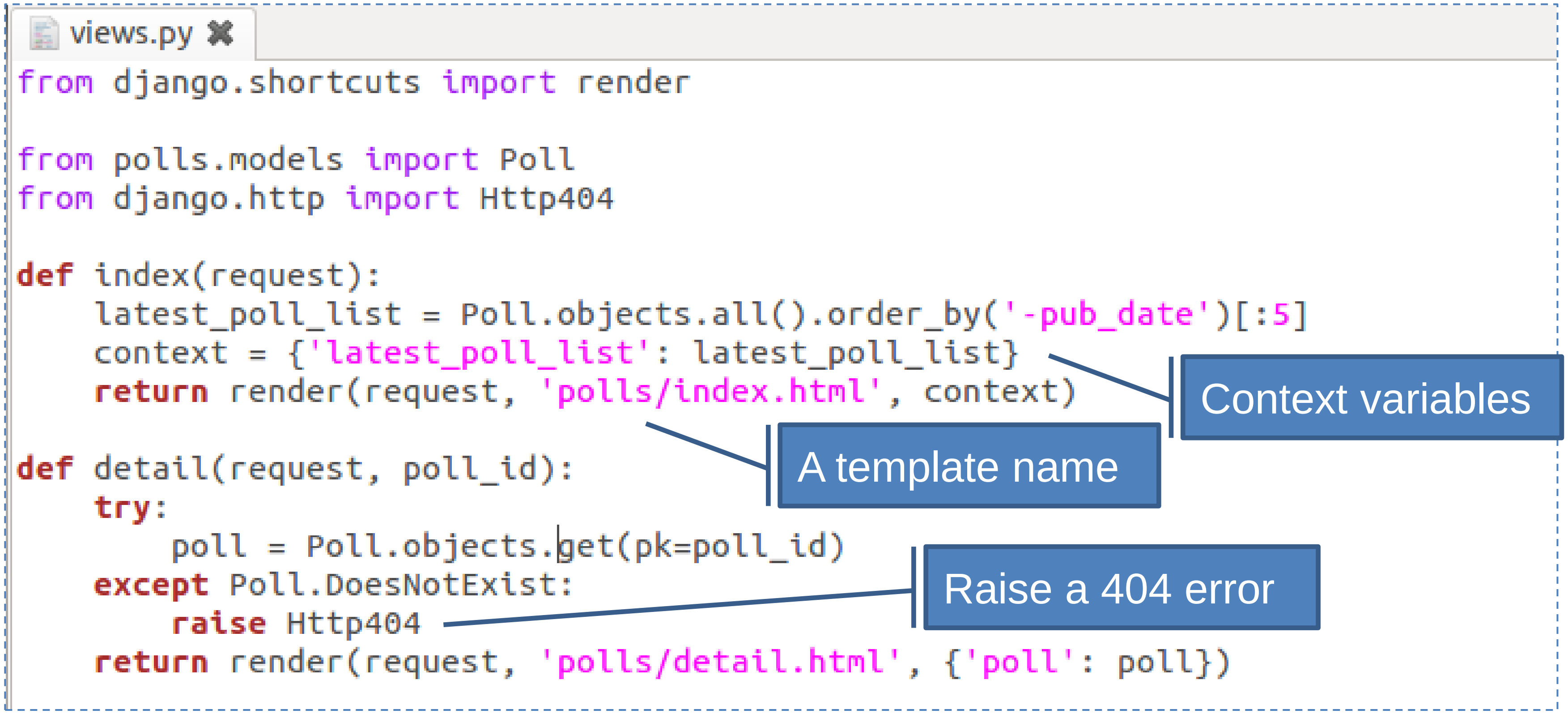
```
{% if latest_poll_list %}
    <ul>
    {% for poll in latest_poll_list %}
        <li><a href="/polls/{{ poll.id }}/">{{ poll.question }}</a></li>
    {% endfor %}
    </ul>
{% else %}
    <p>No polls are available.</p>
{% endif %}
```

- Create a file called **detail.html** and put the following code in that template:



```
<h1>{{ poll.question }}</h1>
<ul>
{% for choice in poll.choice_set.all %}
    <li>{{ choice.choice_text }}</li>
{% endfor %}
</ul>
```

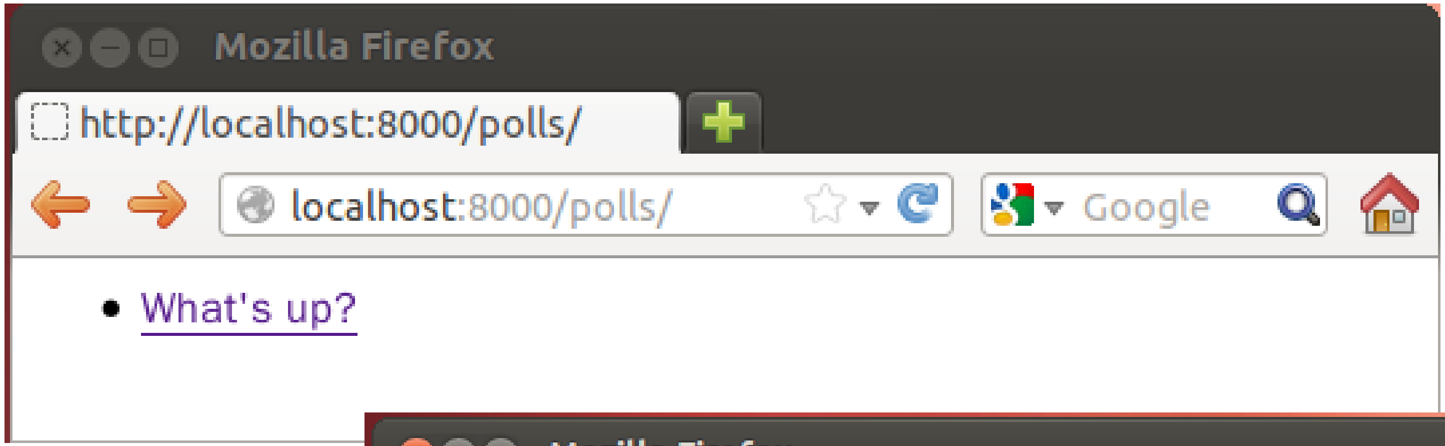
- Open polls/views.py and revise the functions `index` and `detail` as follows:



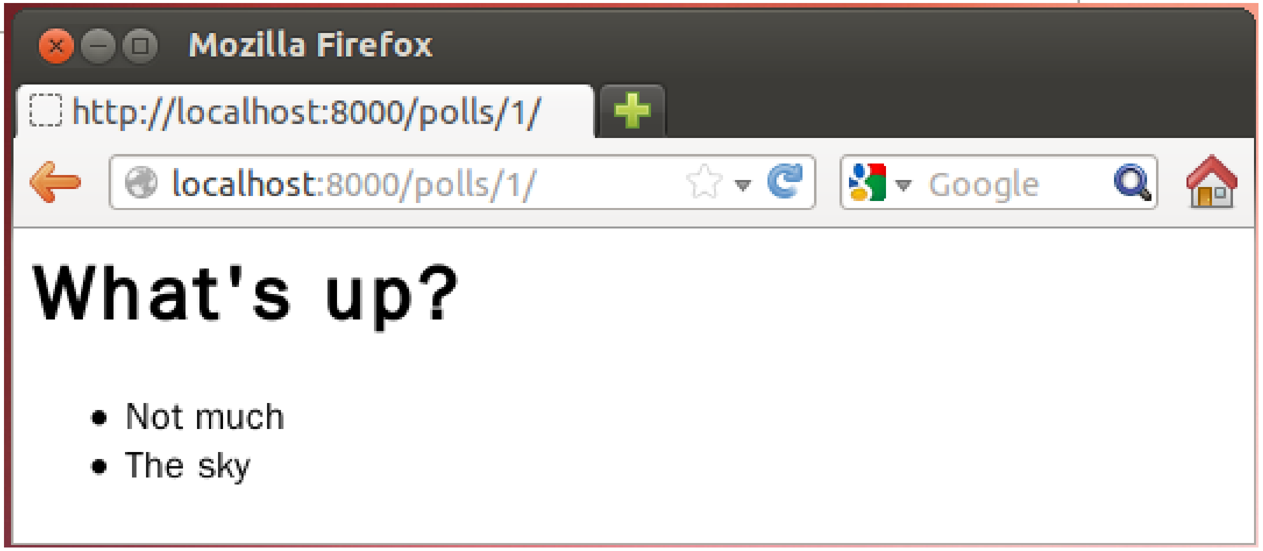
views.py

```
from django.shortcuts import render

from polls.models import Poll
from django.http import Http404

def index(request):
    latest_poll_list = Poll.objects.all().order_by('-pub_date')[:5]
    context = {'latest_poll_list': latest_poll_list}
    return render(request, 'polls/index.html', context)

def detail(request, poll_id):
    try:
        poll = Poll.objects.get(pk=poll_id)
    except Poll.DoesNotExist:
        raise Http404
    return render(request, 'polls/detail.html', {'poll': poll})
```

What You Should See



- What's up?



What's up?

- Not much
- The sky

A shortcut: `render()`

```
from django.http import HttpResponse
from django.template import Context, loader
from polls.models import Poll

def index(request):
    latest_poll_list = Poll.objects.order_by('-pub_date')[:5]
    template = loader.get_template('polls/index.html')
    context = Context({
        'latest_poll_list': latest_poll_list,
    })
    return HttpResponse(template.render(context))
```

All Django wants is that `HttpResponse`.

```
from django.shortcuts import render

from polls.models import Poll

def index(request):
    latest_poll_list = Poll.objects.all().order_by('-pub_date')[:5]
    context = {'latest_poll_list': latest_poll_list}
    return render(request, 'polls/index.html', context)
```

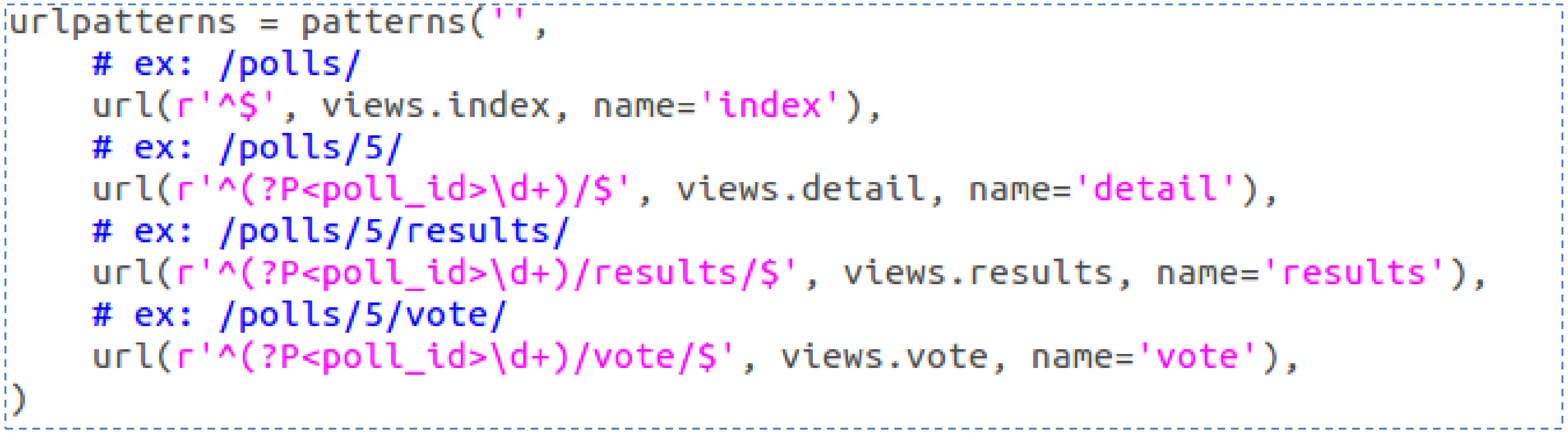
A shortcut: `get_object_or_404()`

```
from django.http import Http404
# ...
def detail(request, poll_id):
    try:
        poll = Poll.objects.get(pk=poll_id)
    except Poll.DoesNotExist:
        raise Http404
    return render(request, 'polls/detail.html', {'poll': poll})
```

```
from django.shortcuts import render, get_object_or_404
# ...
def detail(request, poll_id):
    poll = get_object_or_404(Poll, pk=poll_id)
    return render(request, 'polls/detail.html', {'poll': poll})
```

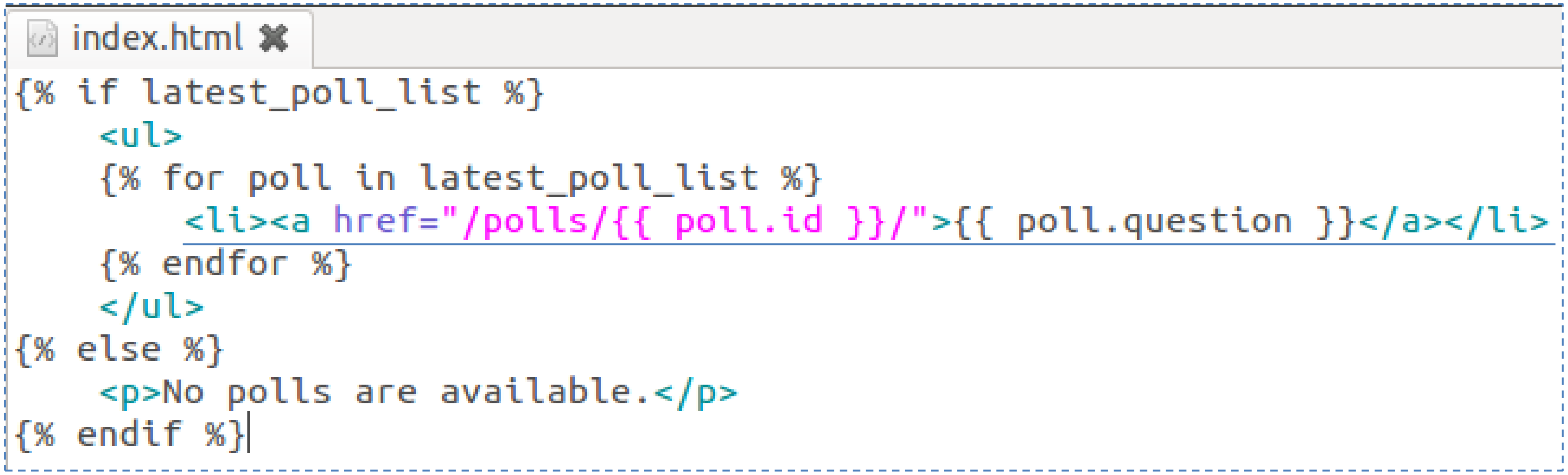
Removing Hardcoded URLs in Templates

- Since you defined the `name` argument in the `url()` functions in the `polls.urls` module…



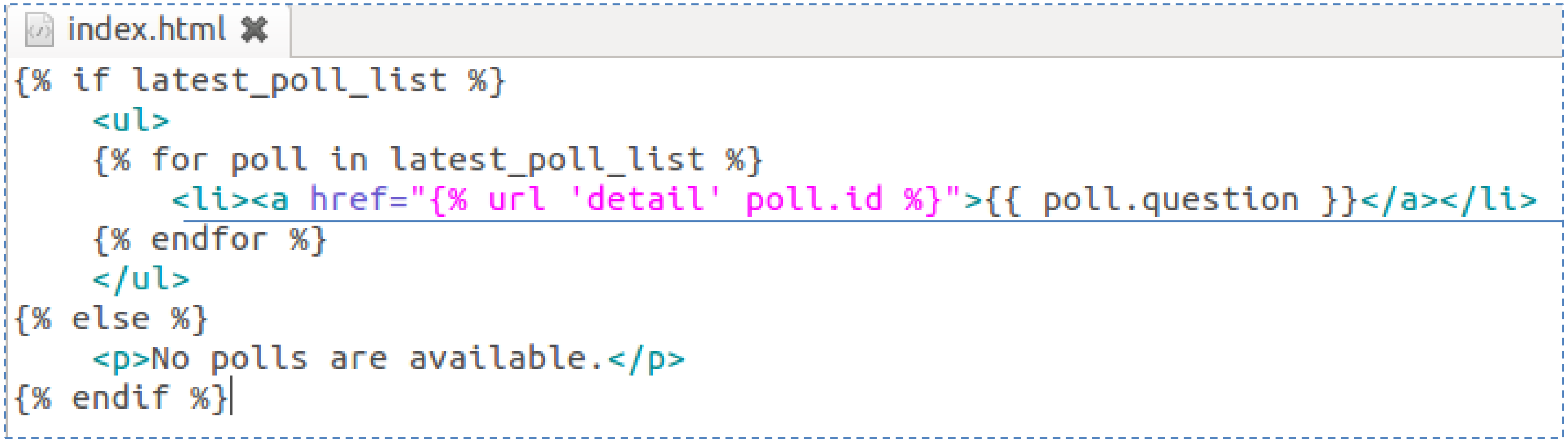
```
urlpatterns = patterns('',
    # ex: /polls/
    url(r'^$', views.index, name='index'),
    # ex: /polls/5/
    url(r'^(?P<poll_id>\d+)/$', views.detail, name='detail'),
    # ex: /polls/5/results/
    url(r'^(?P<poll_id>\d+)/results/$', views.results, name='results'),
    # ex: /polls/5/vote/
    url(r'^(?P<poll_id>\d+)/vote/$', views.vote, name='vote'),
)
```

- You can remove a reliance on specific URL paths…


index.html

```
{% if latest_poll_list %}
    <ul>
    {% for poll in latest_poll_list %}
        <li><a href="/polls/{{ poll.id }}/">{{ poll.question }}</a></li>
    {% endfor %}
    </ul>
{% else %}
    <p>No polls are available.</p>
{% endif %}
```

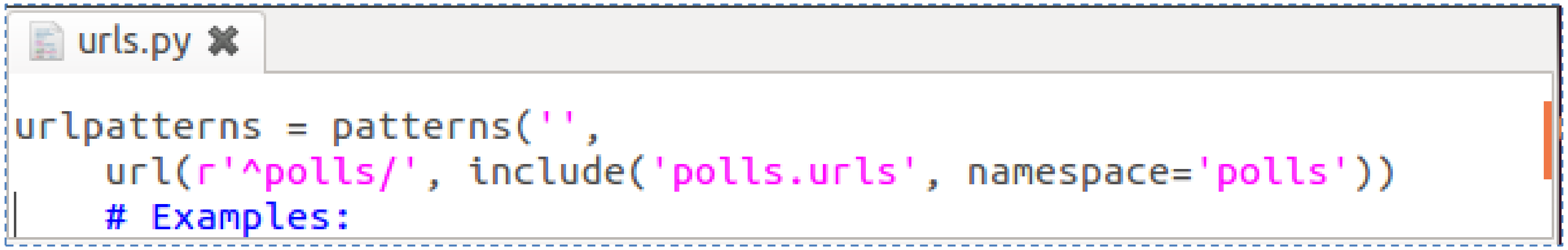
- By using the `{% url %}` template tag:


index.html

```
{% if latest_poll_list %}
    <ul>
    {% for poll in latest_poll_list %}
        <li><a href="{% url 'detail' poll.id %}">{{ poll.question }}</a></li>
    {% endfor %}
    </ul>
{% else %}
    <p>No polls are available.</p>
{% endif %}
```

Namespacing URL Names (Exercise 13)

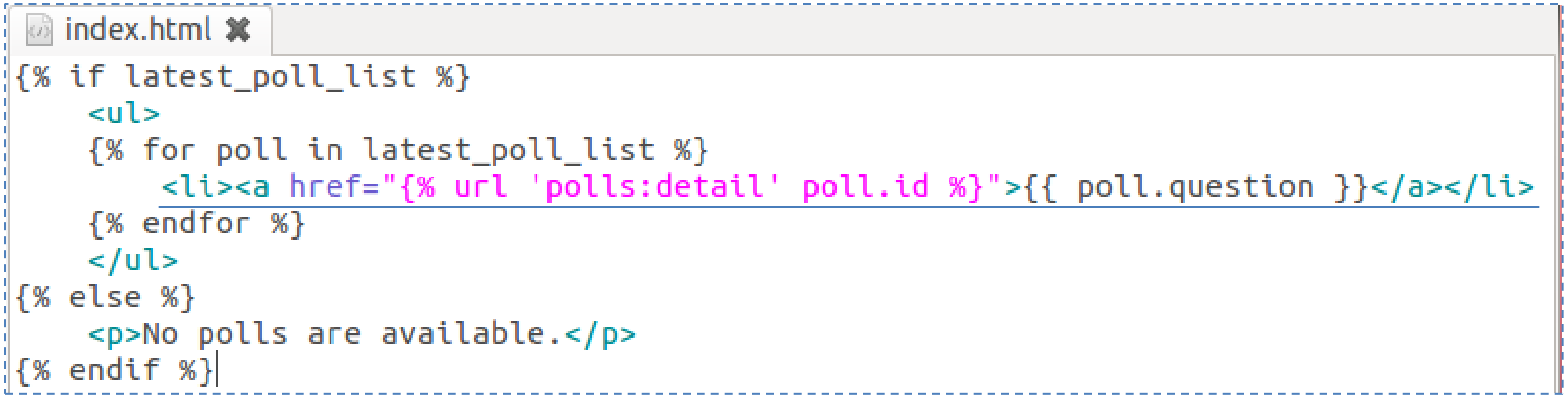
- In the **mysite/urls.py** file, change `url` to `include` namespacing:



urls.py

```
urlpatterns = patterns('',
    url(r'^polls/', include('polls.urls', namespace='polls'))
    # Examples:
```

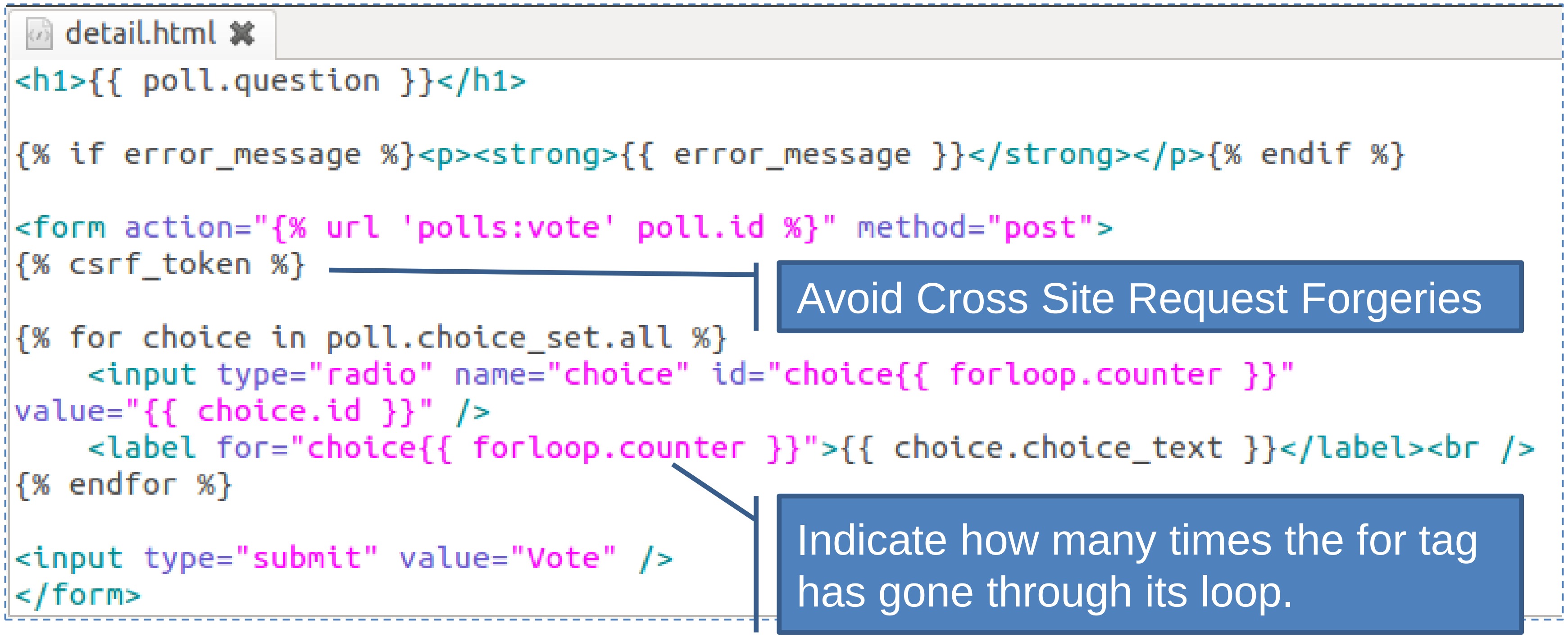
- Change the url of your **polls/index.html** template:



index.html

```
{% if latest_poll_list %}
    <ul>
    {% for poll in latest_poll_list %}
        <li><a href="{% url 'polls:detail' poll.id %}">{{ poll.question }}</a></li>
    {% endfor %}
    </ul>
{% else %}
    <p>No polls are available.</p>
{% endif %}
```

- Update **polls/detail.html** to contains an HTML `<form>` element:



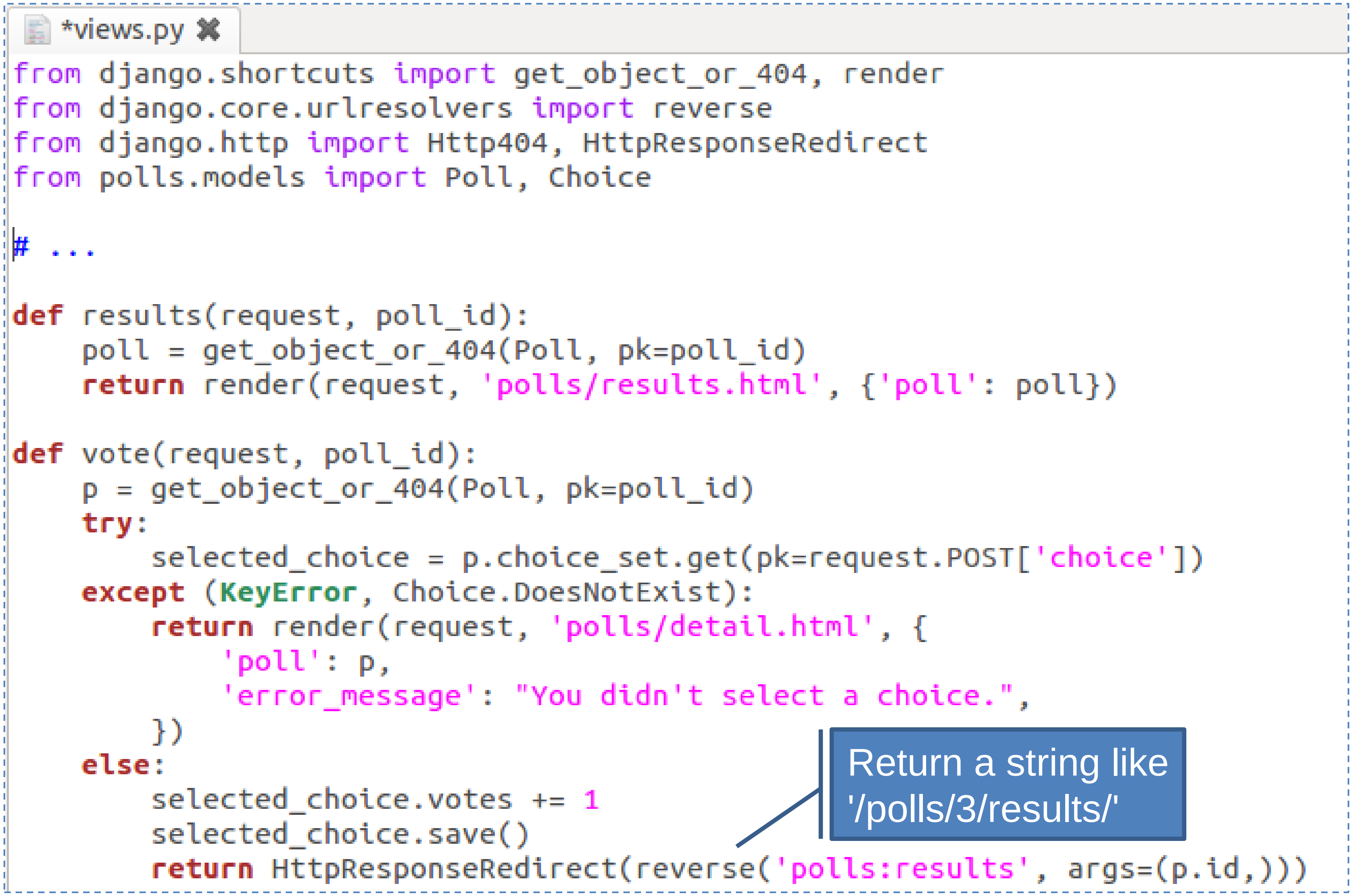
detail.html

```
<h1>{{ poll.question }}</h1>

{% if error_message %}<p><strong>{{ error_message }}</strong></p>{% endif %}

<form action="{% url 'polls:vote' poll.id %}" method="post">
{% csrf_token %}

{% for choice in poll.choice_set.all %}
    <input type="radio" name="choice" id="choice{{ forloop.counter }}"
value="{{ choice.id }}" />
    <label for="choice{{ forloop.counter }}">{{ choice.choice_text }}</label><br />
{% endfor %}

<input type="submit" value="Vote" />
</form>
```

Avoid Cross Site Request Forgeries

Indicate how many times the for tag has gone through its loop.

- Add the following to **polls/views.py**:



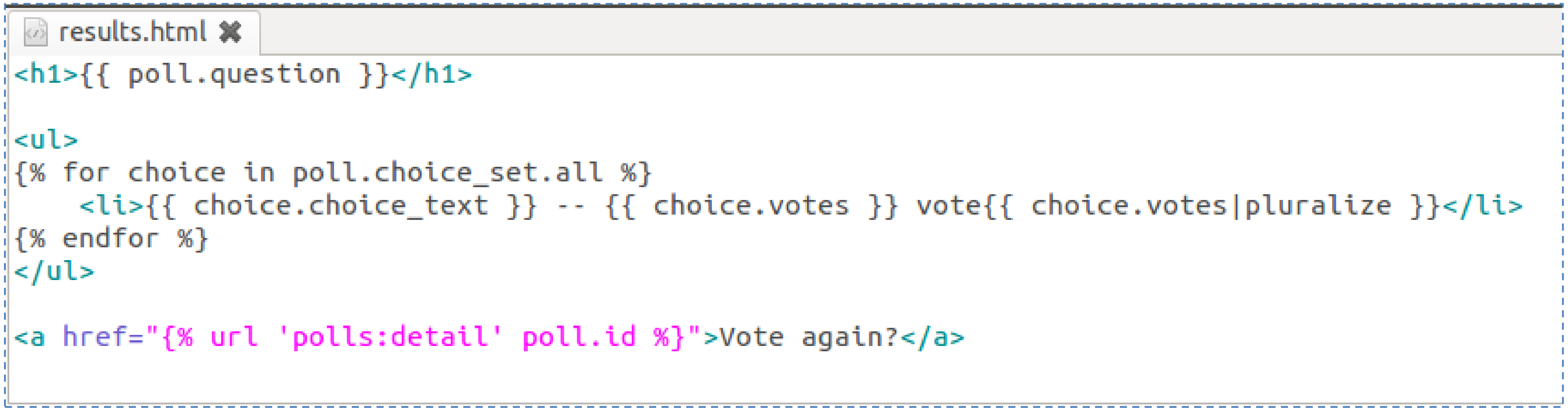
*views.py

```
from django.shortcuts import get_object_or_404, render
from django.core.urlresolvers import reverse
from django.http import Http404, HttpResponseRedirect
from polls.models import Poll, Choice

# ...

def results(request, poll_id):
    poll = get_object_or_404(Poll, pk=poll_id)
    return render(request, 'polls/results.html', {'poll': poll})

def vote(request, poll_id):
    p = get_object_or_404(Poll, pk=poll_id)
    try:
        selected_choice = p.choice_set.get(pk=request.POST['choice'])
    except (KeyError, Choice.DoesNotExist):
        return render(request, 'polls/detail.html', {
            'poll': p,
            'error_message': "You didn't select a choice.",
        })
    else:
        selected_choice.votes += 1
        selected_choice.save()
        return HttpResponseRedirect(reverse('polls:results', args=(p.id,)))
```

Return a string like '/polls/3/results/'

Writing a Simple Form（Exercise 13 Continued）

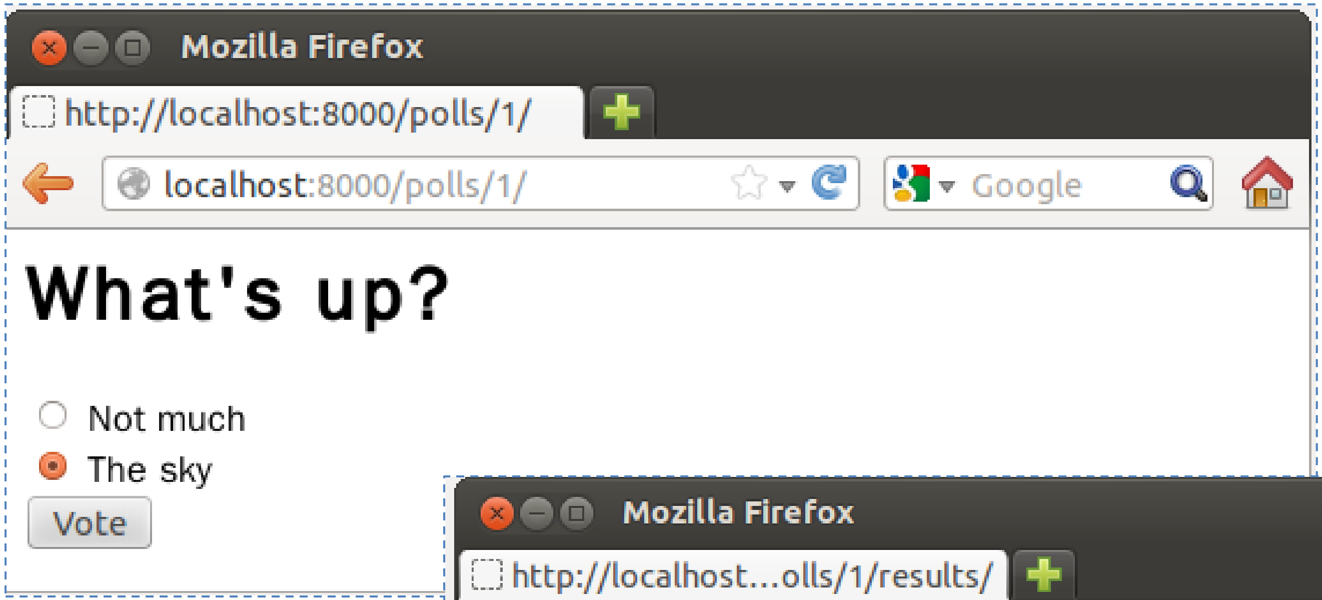
- Create a **polls/results.html** template:



results.html

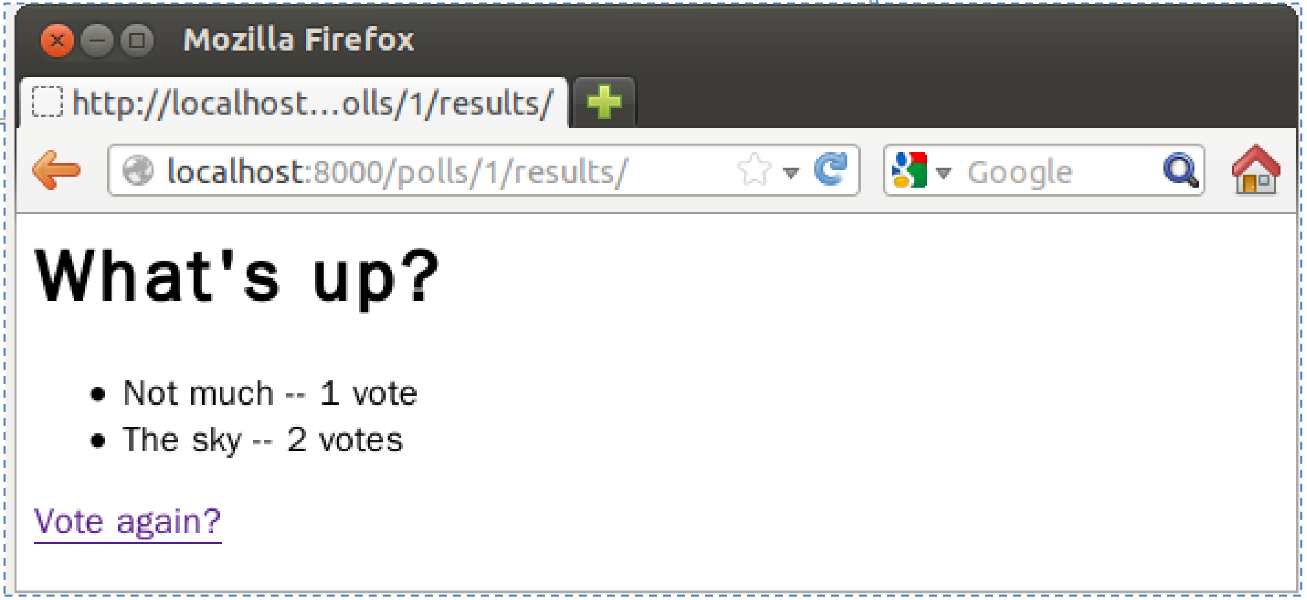
```
<h1>{{ poll.question }}</h1>

<ul>
{% for choice in poll.choice_set.all %}
    <li>{{ choice.choice_text }} -- {{ choice.votes }} vote{{ choice.votes|pluralize }}</li>
{% endfor %}
</ul>

<a href="{% url 'polls:detail' poll.id %}">Vote again?</a>
```

What You Should See



What's up?

( ) Not much
(●) The sky
Vote


Mozilla Firefox
http://localhost...olls/1/results/
localhost:8000/polls/1/results/
Google

What's up?

- Not much -- 1 vote
- The sky -- 2 votes

Vote again?

A Bit About CSRF

- Include malicious code or a link in a page that accesses a web application that the user has authenticated and the session has not timed out.
- A **Cross-Site Request Forgery** Example.
  - Bob’s session at www.webapp.com is still alive.
  - In a message board, Bob views a post from a hacker where there is a crafted HTML image element.

```
<img src="http://www.webapp.com/project/1/destroy">
```

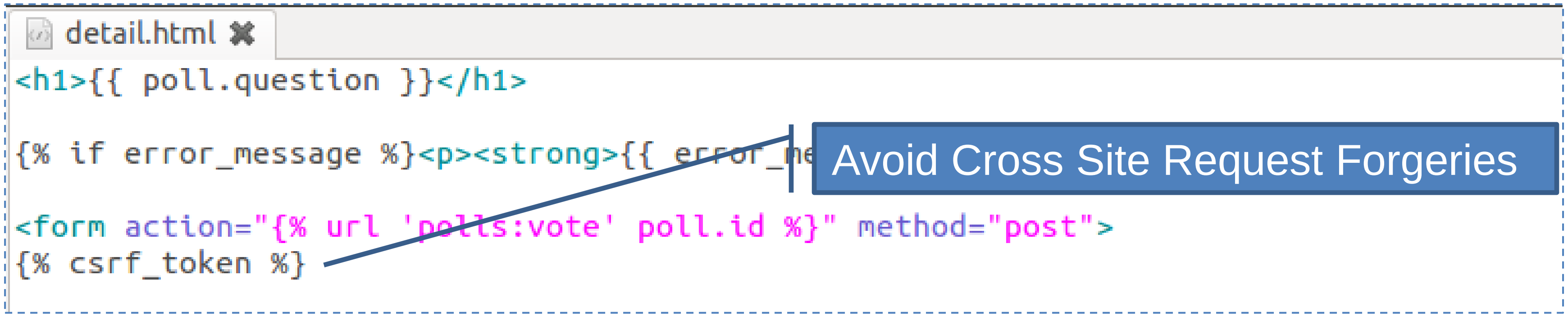
- The actual crafted image or link isn’t necessarily situated in the web application’s domain, it can be anywhere – in a forum, blog post or email.
- POST requests can be sent automatically, too.

```
<a href="http://www.harmless.com/" onclick="
  var f = document.createElement('form');
  f.style.display = 'none';
  this.parentNode.appendChild(f);
  f.method = 'POST';
  f.action = 'http://www.example.com/account/destroy';
  f.submit();
  return false;">To the harmless survey</a>
```

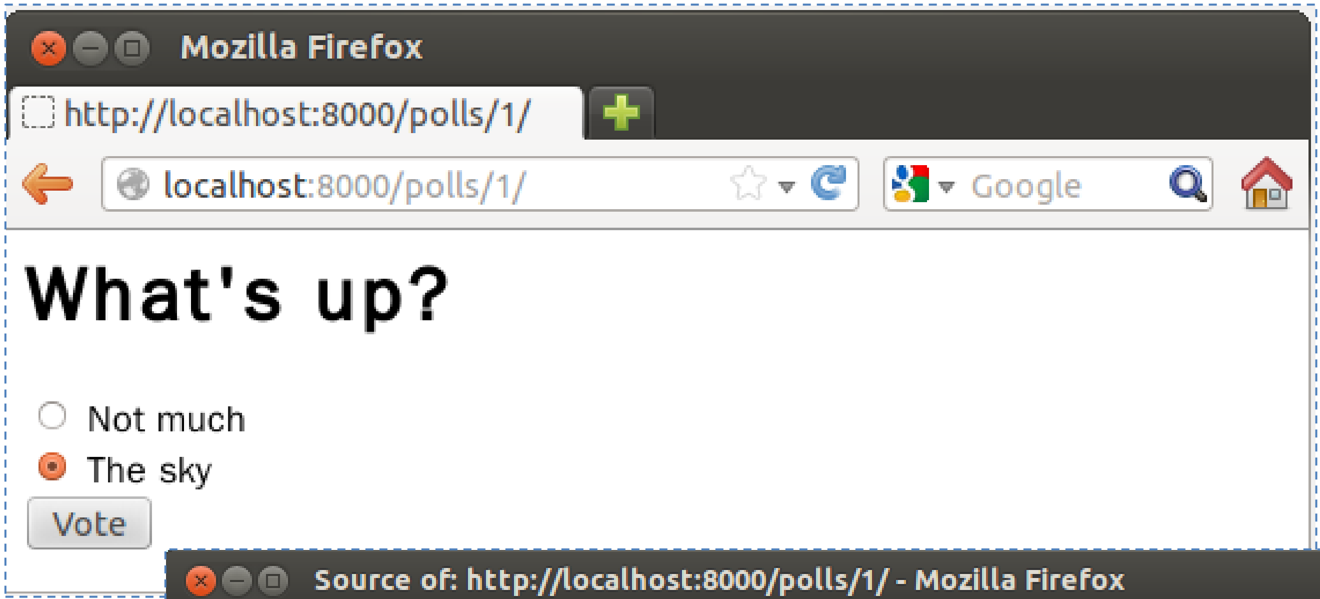
```
<img src="http://www.harmless.com/img" width="400"
height="400" onmouseover="..." />
```

CSRF Countermeasures

- Use **GET** and **POST** appropriately.
  - Use GET if the request is **idempotent**.
  - Use POST if the request changes the **state** of the server.
- Use a security token in non-GET requests.
  - (If your web application is RESTful, you might be used to additional HTTP verbs, such as PUT or DELETE.)



```
<h1>{{ poll.question }}</h1>

{% if error_message %}<p><strong>{{ error_me

<form action="{% url 'polls:vote' poll.id %}" method="post">
{% csrf_token %}
```

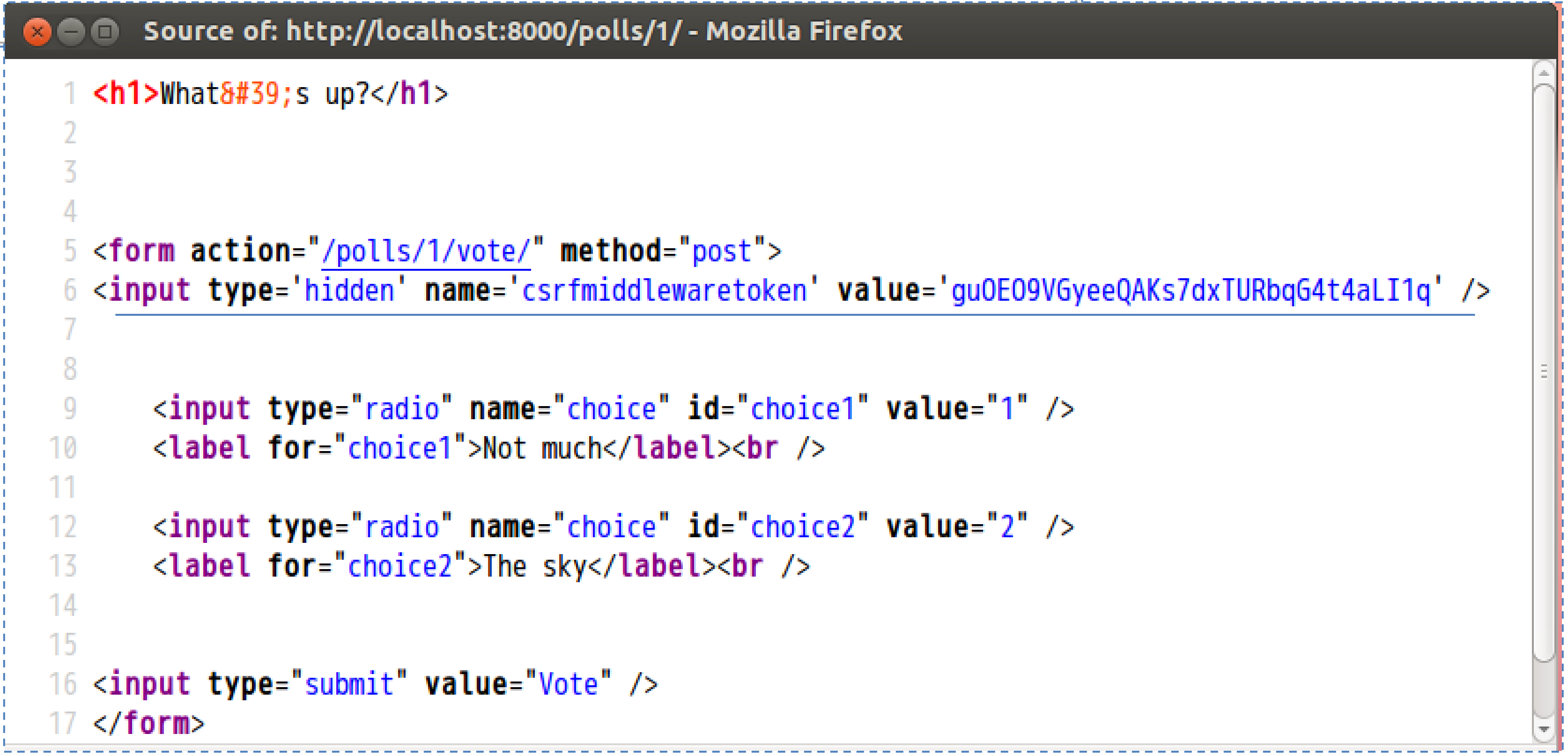


What's up?

Not much

The sky

Vote



```
<h1>What&#39;s up?</h1>



<form action="/polls/1/vote/" method="post">
<input type='hidden' name='csrfmiddlewaretoken' value='guOE09VGyeeQAKs7dxTURbqG4t4aLI1q' />


    <input type="radio" name="choice" id="choice1" value="1" />
    <label for="choice1">Not much</label><br />

    <input type="radio" name="choice" id="choice2" value="2" />
    <label for="choice2">The sky</label><br />


<input type="submit" value="Vote" />
</form>
```

Testing

- The `assert` statement
  - A convenient way to insert debugging assertions into a program.
- The `doctest` module
  - Search for pieces of text that look like interactive sessions, and then executes them to verify that they work exactly as shown.
- The `unittest` module
  - Sometimes referred to as “PyUnit,” is a Python language version of JUnit.
- Third-party testing tools
  - nose (https://nose.readthedocs.org/en/latest/)
  - pytest (http://pytest.org)

Before we go on…

- Within a module, the module's name (as a string) is available as the value of the global variable `__name__`.
- When you run a Python module with:

```
python fibo.py <arguments>
```

- The code in the module will be executed, just as if you imported it, but with the `__name__` set to `'__main__'`.
- This means that you can include a self-test at the end of the module:

```
if __name__ == "__main__":
    self_test_code_here
```

assert

- A convenient way to insert assertions into a program:

```
assert_stmt ::=  "assert" expression ["," expression]
```

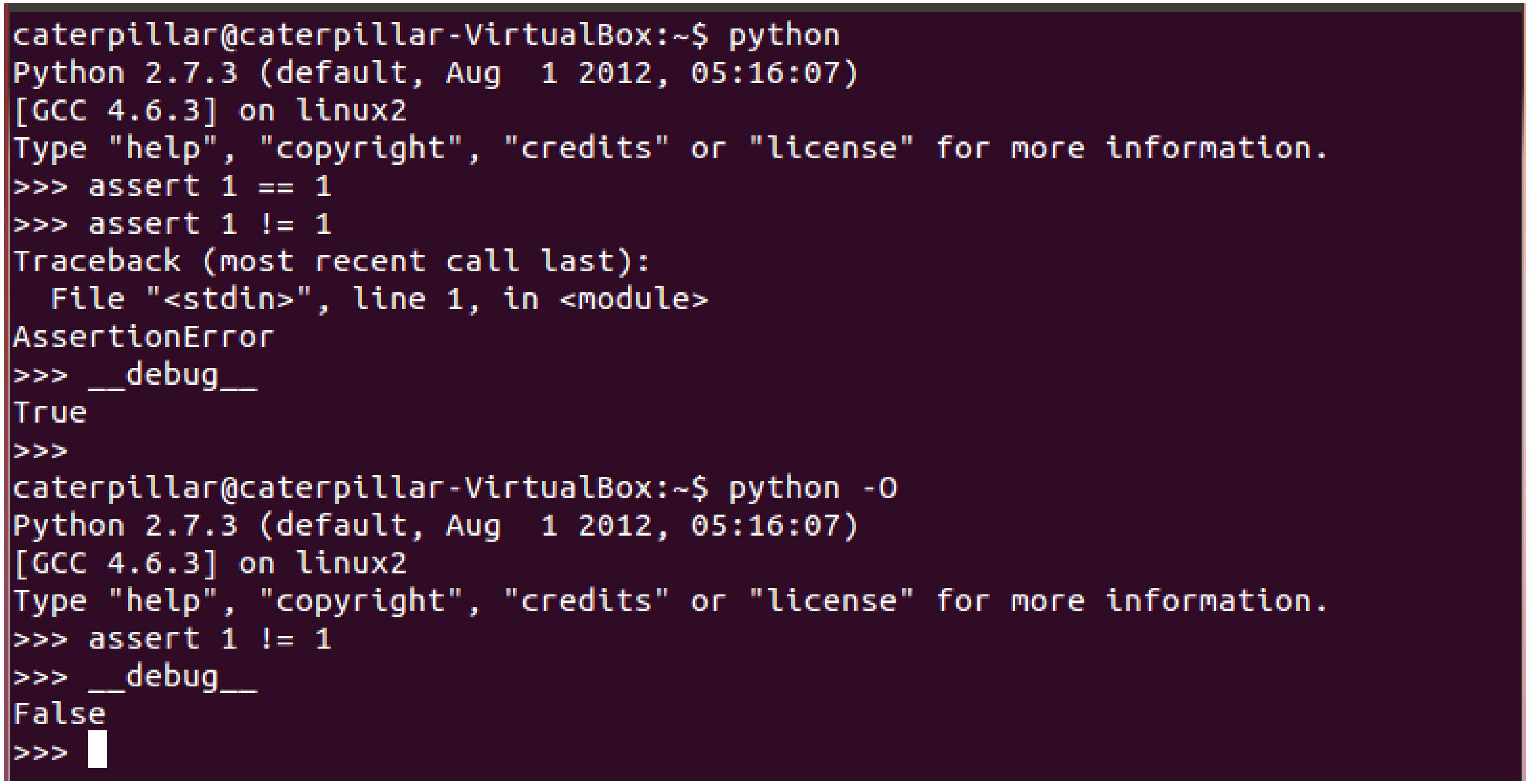
- The `assert expression` is equivalent to:

```
if __debug__:
   if not expression: raise AssertionError
```

- The `assert expression1, expression2` is equivalent to:

```
if __debug__:
   if not expression1: raise AssertionError(expression2)
```

- The built-in variable `__debug__` is `True` under normal circumstances, `False` when optimization is requested (command line option `-O`).



```
caterpillar@caterpillar-VirtualBox:~$ python
Python 2.7.3 (default, Aug  1 2012, 05:16:07)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> assert 1 == 1
>>> assert 1 != 1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AssertionError
>>> __debug__
True
>>>
caterpillar@caterpillar-VirtualBox:~$ python -O
Python 2.7.3 (default, Aug  1 2012, 05:16:07)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> assert 1 != 1
>>> __debug__
False
>>> 
```

When to Use Assertions?

- Preconditions (in private functions only)
  - The requirements which a function requires its caller to fulfill.
- Postconditions
  - Verifing the promises made by a function to its caller.
- Class invariants
  - Validating object state.
- Internal Invariants
  - Using assertions instead of comments.
- Unreachable code (Control-Flow Invariants)
  - Parts of your program which you expect to be unreachable.

Preconditions

- An Example:

Defensive Programming

```
def __set_refresh_Interval(interval):
    if interval > 0 and interval <= 1000 / MAX_REFRESH_RATE:
        raise ValueError('Illegal interval: ' + interval)
    # set the refresh interval or others ...
```

```
def __set_refresh_Interval(rate):
    (assert interval > 0 and interval <= 1000 / MAX_REFRESH_RATE,
            'Illegal interval: ' + interval)
    # set the refresh interval or others ...
```

Internal Invariants

```
if balance > 10000:
    ...
else if 10000 > balance > 100:
    ...
else: # the balance should be less than 100
    ...
```

An assumption concerning a program's behavior

```
if balance > 10000:
    ...
else if 10000 > balance >= 100:
    ...
else:
    assert balance < 100, balance
```

Unreachable code

- An example:

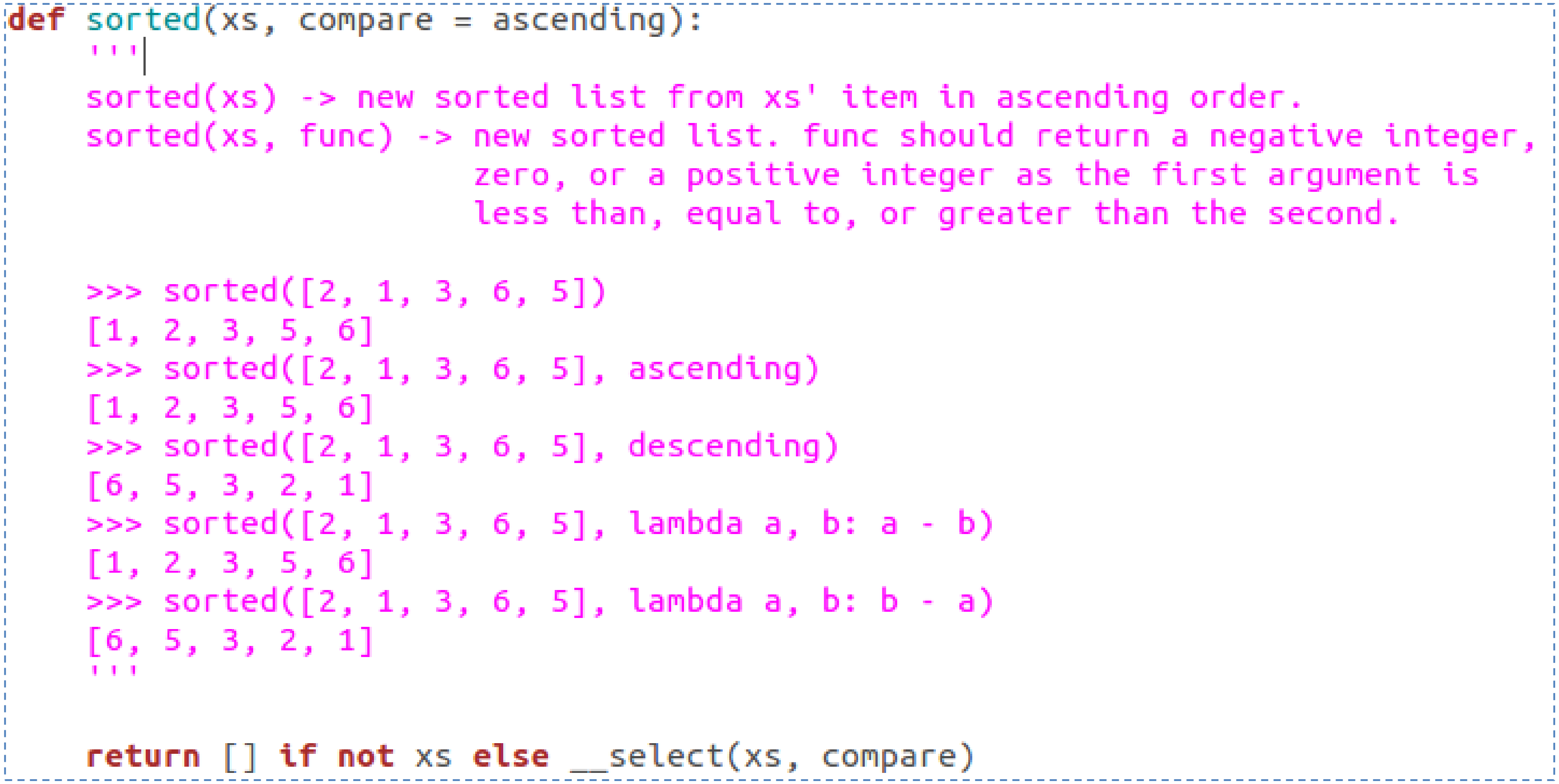
```
def foo(list):
    for ele in list:
        if ...:
            return
    # execution should never reach this point!!!
```

```
def foo(list):
    for ele in list:
        if ...:
            return
    assert False
```

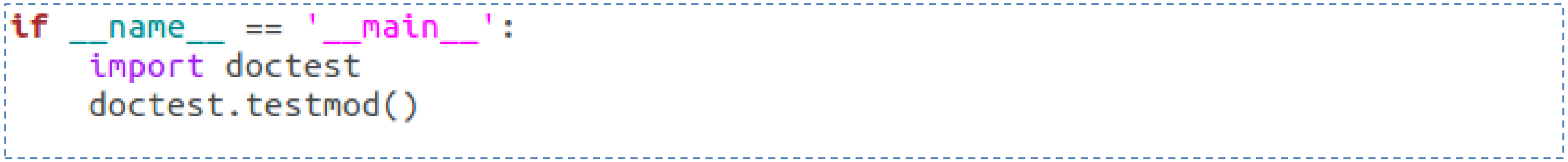
doctest

- Checks that a module's **docstrings** are up-to-date.
- Performs regression testing by verifying that interactive examples from a test.
- Writes tutorial documentation for a package, liberally illustrated with input-output examples. This has the flavor of **"literate testing"** or **"executable documentation"**.

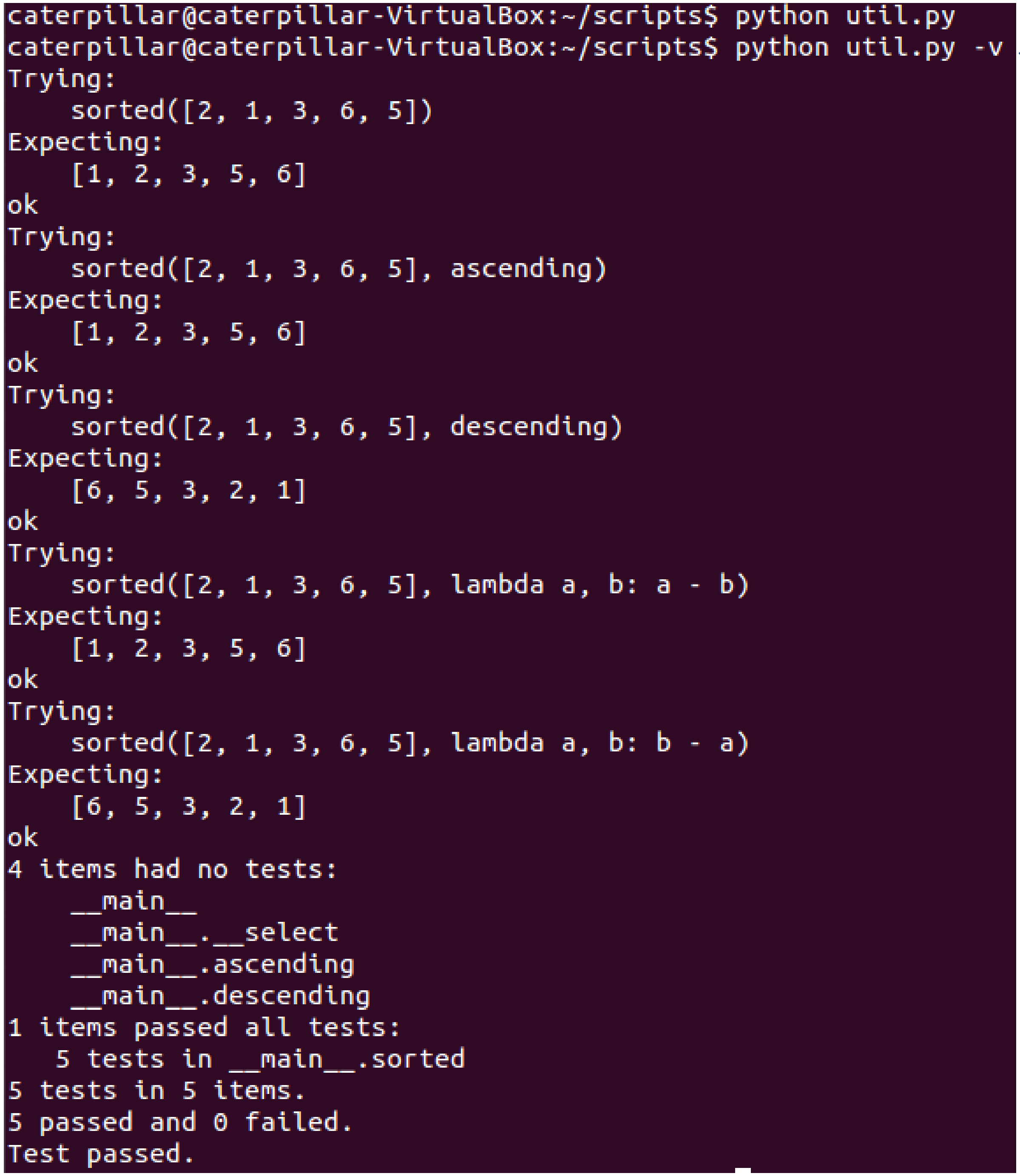
Checking Examples in Docstrings



```python
def sorted(xs, compare = ascending):
    '''
    sorted(xs) -> new sorted list from xs' item in ascending order.
    sorted(xs, func) -> new sorted list. func should return a negative integer,
                        zero, or a positive integer as the first argument is
                        less than, equal to, or greater than the second.

    >>> sorted([2, 1, 3, 6, 5])
    [1, 2, 3, 5, 6]
    >>> sorted([2, 1, 3, 6, 5], ascending)
    [1, 2, 3, 5, 6]
    >>> sorted([2, 1, 3, 6, 5], descending)
    [6, 5, 3, 2, 1]
    >>> sorted([2, 1, 3, 6, 5], lambda a, b: a - b)
    [1, 2, 3, 5, 6]
    >>> sorted([2, 1, 3, 6, 5], lambda a, b: b - a)
    [6, 5, 3, 2, 1]
    '''

    return [] if not xs else __select(xs, compare)
```



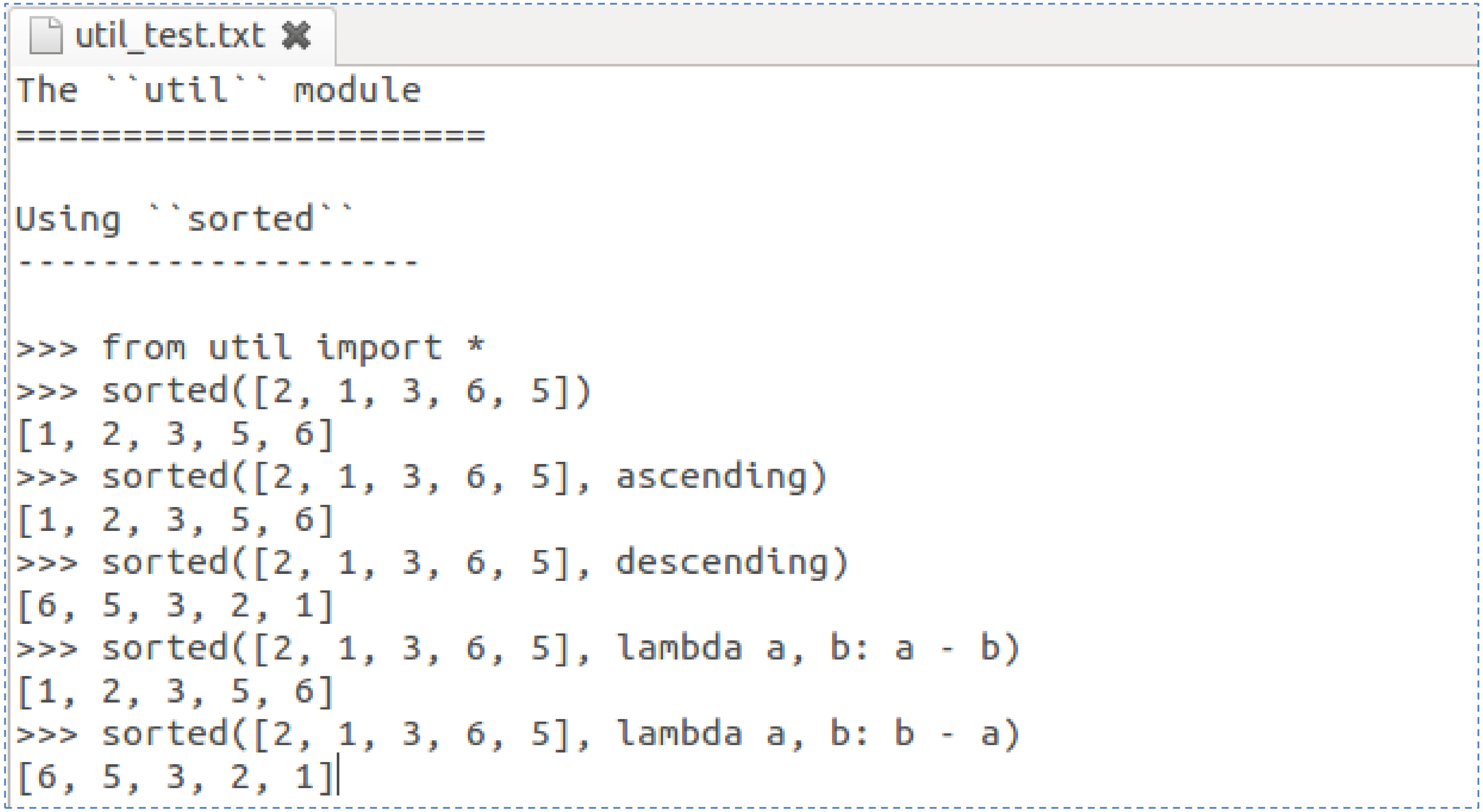
```python
if __name__ == '__main__':
    import doctest
    doctest.testmod()
```



```
caterpillar@caterpillar-VirtualBox:~/scripts$ python util.py
caterpillar@caterpillar-VirtualBox:~/scripts$ python util.py -v
Trying:
    sorted([2, 1, 3, 6, 5])
Expecting:
    [1, 2, 3, 5, 6]
ok
Trying:
    sorted([2, 1, 3, 6, 5], ascending)
Expecting:
    [1, 2, 3, 5, 6]
ok
Trying:
    sorted([2, 1, 3, 6, 5], descending)
Expecting:
    [6, 5, 3, 2, 1]
ok
Trying:
    sorted([2, 1, 3, 6, 5], lambda a, b: a - b)
Expecting:
    [1, 2, 3, 5, 6]
ok
Trying:
    sorted([2, 1, 3, 6, 5], lambda a, b: b - a)
Expecting:
    [6, 5, 3, 2, 1]
ok
4 items had no tests:
    __main__
    __main__.__select
    __main__.ascending
    __main__.descending
1 items passed all tests:
   5 tests in __main__.sorted
5 tests in 5 items.
5 passed and 0 failed.
Test passed.
```

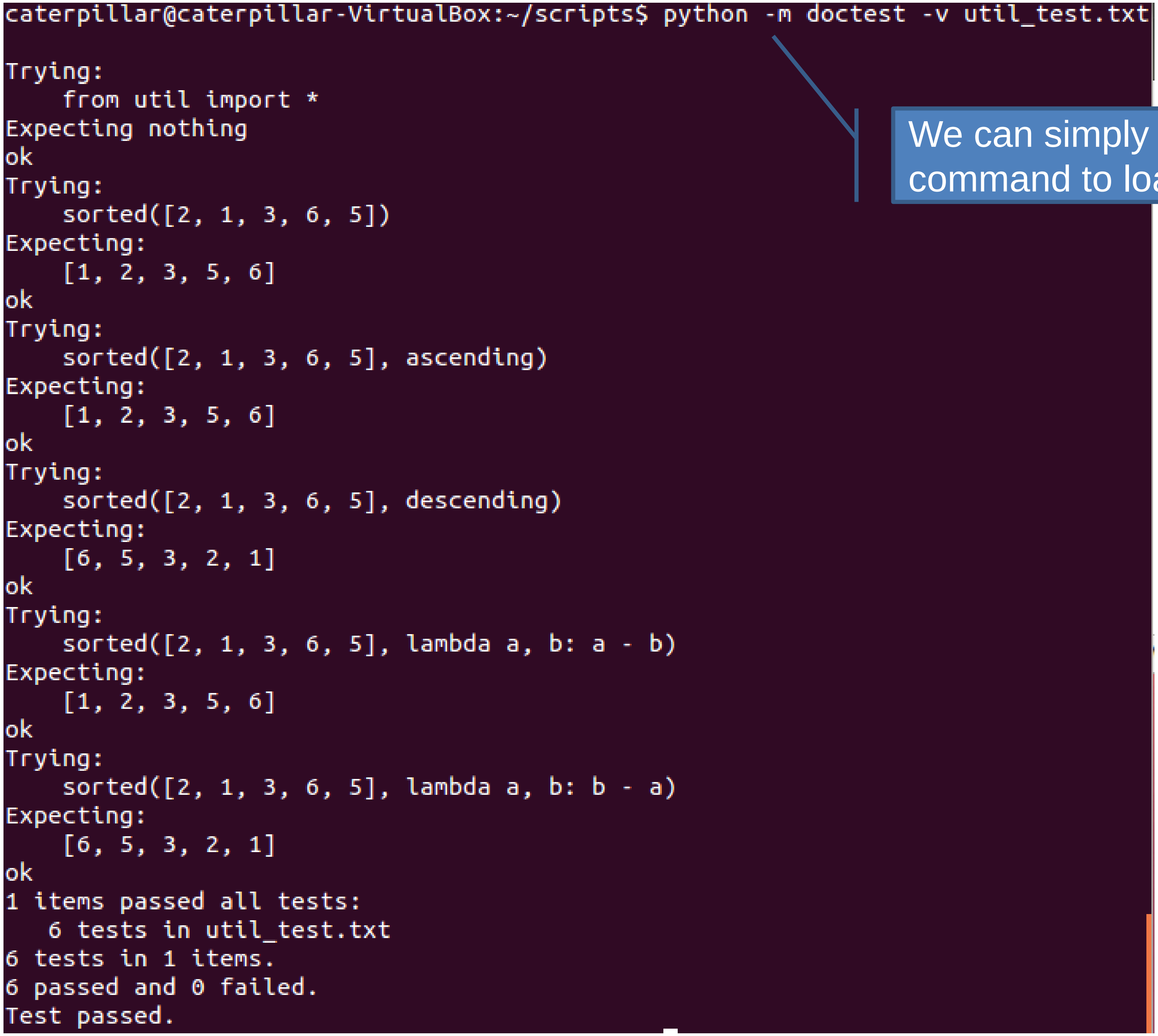
Print a detailed log.

Checking Examples in a Text File



```
The ``util`` module
======================

Using ``sorted``
--------------------

>>> from util import *
>>> sorted([2, 1, 3, 6, 5])
[1, 2, 3, 5, 6]
>>> sorted([2, 1, 3, 6, 5], ascending)
[1, 2, 3, 5, 6]
>>> sorted([2, 1, 3, 6, 5], descending)
[6, 5, 3, 2, 1]
>>> sorted([2, 1, 3, 6, 5], lambda a, b: a - b)
[1, 2, 3, 5, 6]
>>> sorted([2, 1, 3, 6, 5], lambda a, b: b - a)
[6, 5, 3, 2, 1]
```

```
import doctest
doctest.testfile(“util_test.txt")
```



```
caterpillar@caterpillar-VirtualBox:~/scripts$ python -m doctest -v util_test.txt

Trying:
    from util import *
Expecting nothing
ok
Trying:
    sorted([2, 1, 3, 6, 5])
Expecting:
    [1, 2, 3, 5, 6]
ok
Trying:
    sorted([2, 1, 3, 6, 5], ascending)
Expecting:
    [1, 2, 3, 5, 6]
ok
Trying:
    sorted([2, 1, 3, 6, 5], descending)
Expecting:
    [6, 5, 3, 2, 1]
ok
Trying:
    sorted([2, 1, 3, 6, 5], lambda a, b: a - b)
Expecting:
    [1, 2, 3, 5, 6]
ok
Trying:
    sorted([2, 1, 3, 6, 5], lambda a, b: b - a)
Expecting:
    [6, 5, 3, 2, 1]
ok
1 items passed all tests:
   6 tests in util_test.txt
6 tests in 1 items.
6 passed and 0 failed.
Test passed.
```

We can simply type this command to load a test file.

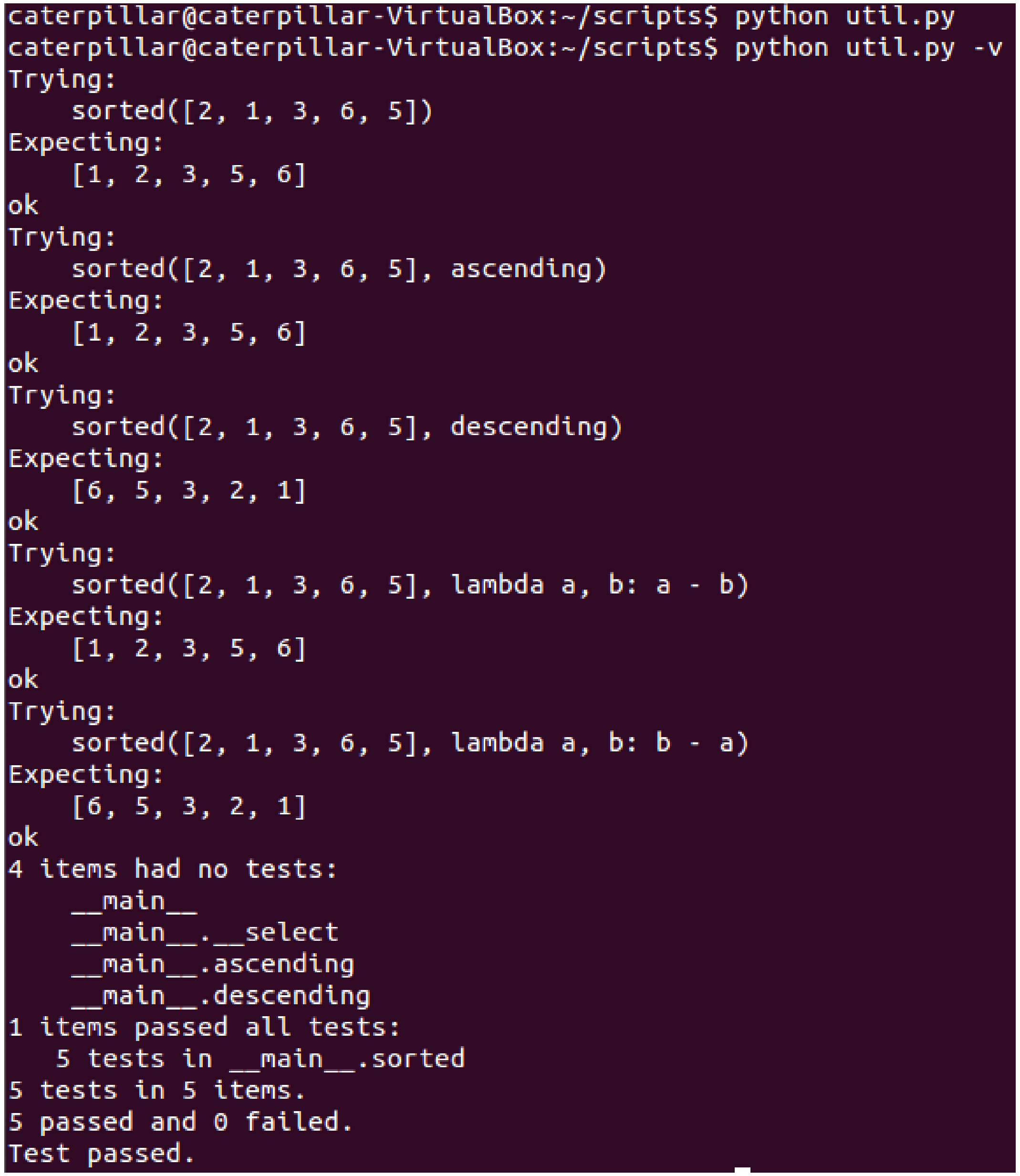
Exercise 14

- Pick up **util.py** located in the **exercises/exercise14** of the lab file. Replace those two `print` statement with the following:

```
if __name__ == '__main__':
    import doctest
    doctest.testmod()
```

- Write docstrings as you seen in the slide of “Checking Examples in Docstrings”.
- Run the following commands and see what happens.
  - `python util.py`
  - `python util.py -v`

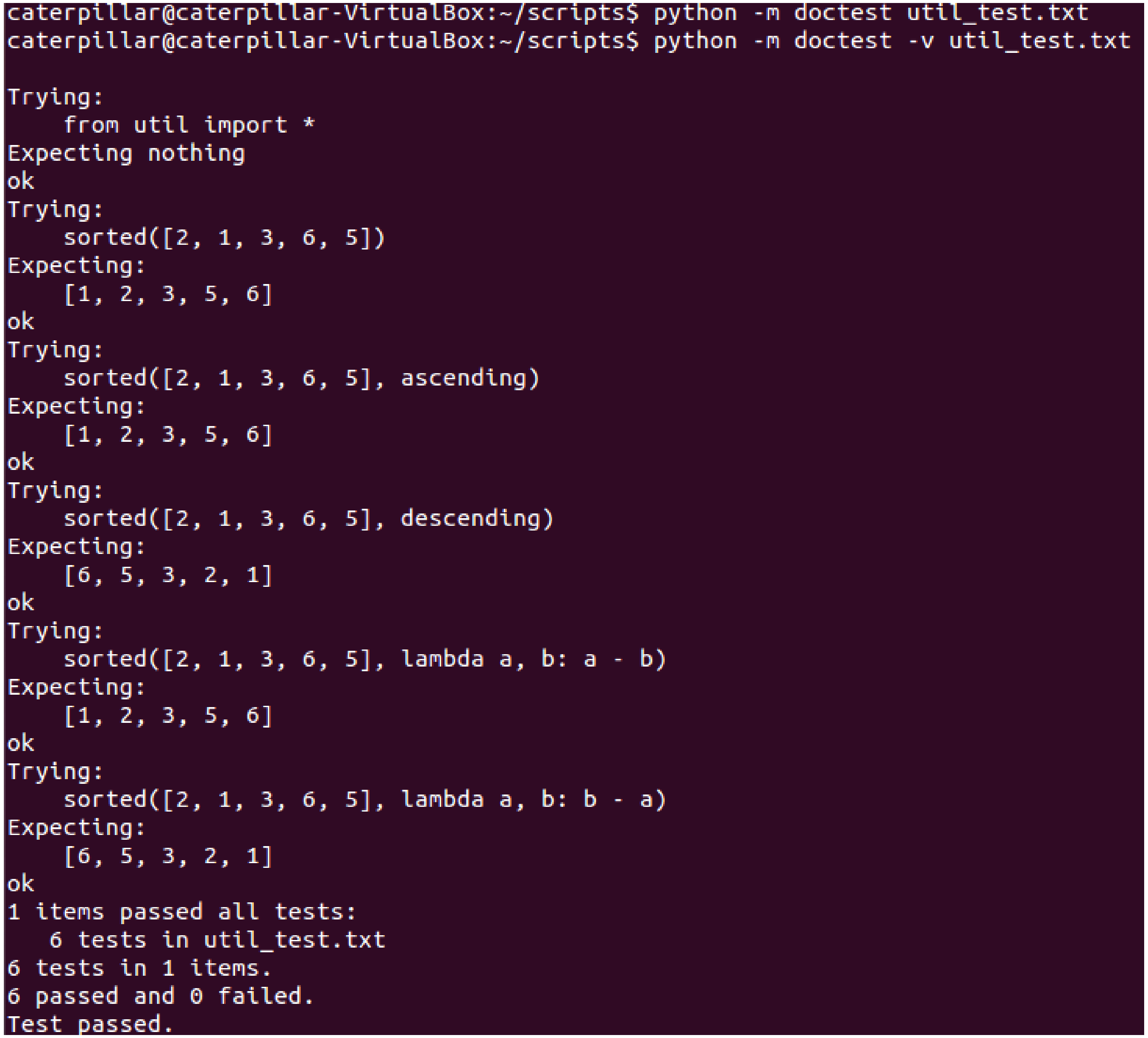
What You Should See



```
caterpillar@caterpillar-VirtualBox:~/scripts$ python util.py
caterpillar@caterpillar-VirtualBox:~/scripts$ python util.py -v
Trying:
    sorted([2, 1, 3, 6, 5])
Expecting:
    [1, 2, 3, 5, 6]
ok
Trying:
    sorted([2, 1, 3, 6, 5], ascending)
Expecting:
    [1, 2, 3, 5, 6]
ok
Trying:
    sorted([2, 1, 3, 6, 5], descending)
Expecting:
    [6, 5, 3, 2, 1]
ok
Trying:
    sorted([2, 1, 3, 6, 5], lambda a, b: a - b)
Expecting:
    [1, 2, 3, 5, 6]
ok
Trying:
    sorted([2, 1, 3, 6, 5], lambda a, b: b - a)
Expecting:
    [6, 5, 3, 2, 1]
ok
4 items had no tests:
    __main__
    __main__.__select
    __main__.ascending
    __main__.descending
1 items passed all tests:
   5 tests in __main__.sorted
5 tests in 5 items.
5 passed and 0 failed.
Test passed.
```

- Edit a text file ‘util_text.txt’ as you see in the slide of “Checking Examples in a Text File”.
- Run the following commands and see what happens.
  - `python -m doctest util_test.txt`
  - `python -m doctest -v util_test.txt`

What You Should See



```
caterpillar@caterpillar-VirtualBox:~/scripts$ python -m doctest util_test.txt
caterpillar@caterpillar-VirtualBox:~/scripts$ python -m doctest -v util_test.txt

Trying:
    from util import *
Expecting nothing
ok
Trying:
    sorted([2, 1, 3, 6, 5])
Expecting:
    [1, 2, 3, 5, 6]
ok
Trying:
    sorted([2, 1, 3, 6, 5], ascending)
Expecting:
    [1, 2, 3, 5, 6]
ok
Trying:
    sorted([2, 1, 3, 6, 5], descending)
Expecting:
    [6, 5, 3, 2, 1]
ok
Trying:
    sorted([2, 1, 3, 6, 5], lambda a, b: a - b)
Expecting:
    [1, 2, 3, 5, 6]
ok
Trying:
    sorted([2, 1, 3, 6, 5], lambda a, b: b - a)
Expecting:
    [6, 5, 3, 2, 1]
ok
1 items passed all tests:
   6 tests in util_test.txt
6 tests in 1 items.
6 passed and 0 failed.
Test passed.
```

References

- Using the Template System
  - https://docs.djangoproject.com/en/1.5/intro/tutorial04/
  - https://docs.djangoproject.com/en/1.5/topics/templates/
- A Bit About Cross-Site Request Forgery
  - http://guides.rubyonrails.org/security.html#cross-site-request-forgery-csrf
- Testing
  - http://docs.python.org/2/tutorial/modules.html
  - http://docs.python.org/2/reference/simple_stmts.html#the-assert-statement
  - http://docs.python.org/2/library/constants.html#__debug__
  - http://docs.oracle.com/javase/1.4.2/docs/guide/lang/assert.html
  - http://docs.python.org/2/library/doctest.html

`unittest` (Testing Continued)

- Test case
  - The smallest unit of testing.
- Test fixture
  - Represents the preparation needed to perform one or more tests, and any associate cleanup actions.
- Test suite
  - A collection of test cases, test suites, or both.
- Test runner
  - A component which orchestrates the execution of tests and provides the outcome to the user.

Test Case

- `unittest` provides a base class, `TestCase`, which may be used to create new test cases.

```
import unittest
import calculator

class CalculatorTestCase(unittest.TestCase):
    def setUp(self):
        self.args = (3, 2)

    def tearDown(self):
        self.args = None

    def test_plus(self):
        expected = 5;
        result = calculator.plus(*self.args);
        self.assertEquals(expected, result);

    def test_minus(self):
        expected = 1;
        result = calculator.minus(*self.args);
        self.assertEquals(expected, result);
```

The individual test is defined with a method whose name starts with `test`.

Test Fixture

- Often, many small test cases will use the same fixture.
- The test runner will run `setUp` prior to each test and invoke `tearDown` after each test.
  - One real case is creating a new table and inserting data in `setUp`, running a test, and then dropping the table in `tearDown`.

Test Suite

- Add specified tests

```
suite = unittest.TestSuite()
suite.addTest(CalculatorTestCase('test_plus'))
suite.addTest(CalculatorTestCase('test_minus'))
```

```
tests = ['test_plus', 'test_minus']
suite = unittest.TestSuite(map(CalculatorTestCase, tests))
```

- Create a test suite and populate it with all tests of a test case automatically.

```
unittest.TestLoader().loadTestsFromTestCase(CalculatorTestCase)
```

- Add one test suite to a test suite.

```
suite2 = unittest.TestSuite()
suite2.addTest(suite)
suite2.addTest(OtherTestCase('test_orz'))
```

- Compose all suites.

```
suite1 = module1.TheTestSuite()
suite2 = module2.TheTestSuite()
alltests = unittest.TestSuite([suite1, suite2])
```

- So, you can compose tests freely.

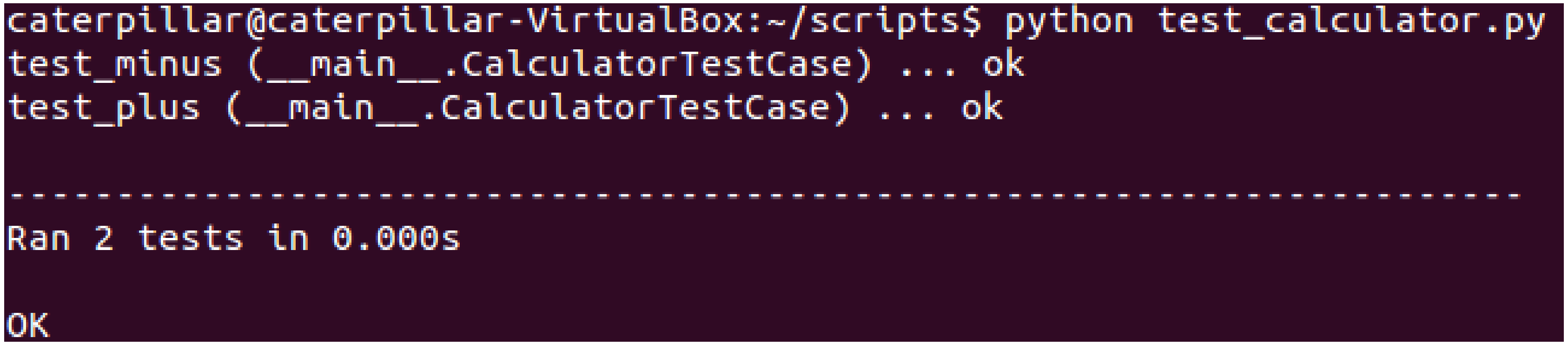
Test Runner

- Use `TextTestRunner` directly.

```
suite = (unittest.TestLoader()
                    .loadTestsFromTestCase(CalculatorTestCase))
unittest.TextTestRunner(verbosity=2).run(suite)
```

- Or…

```
unittest.main(verbosity=2)
```



```
caterpillar@caterpillar-VirtualBox:~/scripts$ python test_calculator.py
test_minus (__main__.CalculatorTestCase) ... ok
test_plus (__main__.CalculatorTestCase) ... ok

----------------------------------------------------------------------
Ran 2 tests in 0.000s

OK
```

Command-Line Interface

- Run tests from modules, classes or even individual test methods:

```
python -m unittest test_module1 test_module2
python -m unittest test_module.TestClass
python -m unittest test_module.TestClass.test_method
```

- Run tests with higher verbosity by passing in the `-v` flag:

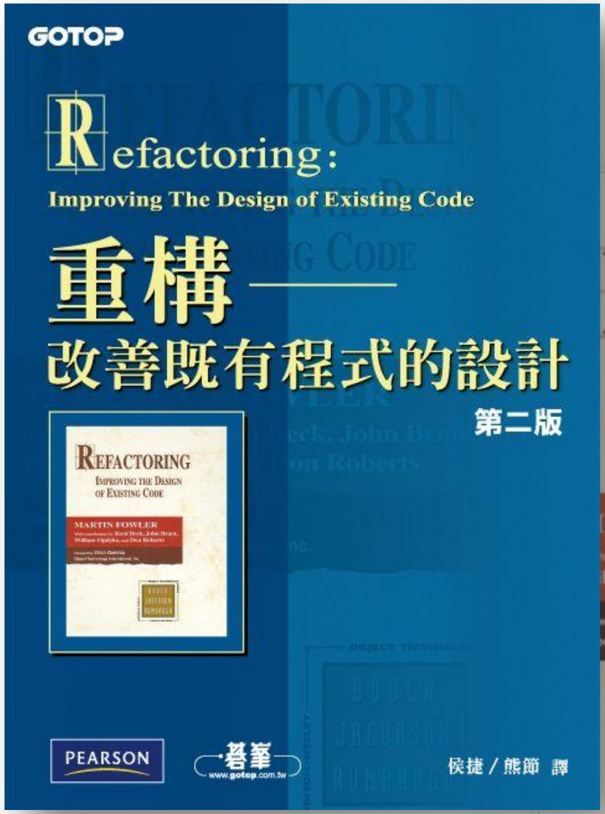
```
python -m unittest -v test_module
```

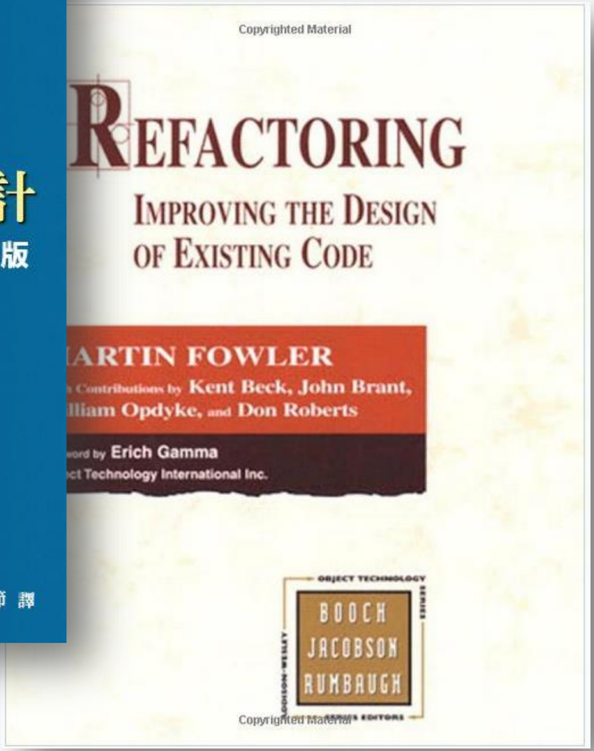
- For a list of all the command-line options:

```
python -m unittest -h
```

Exercise 15

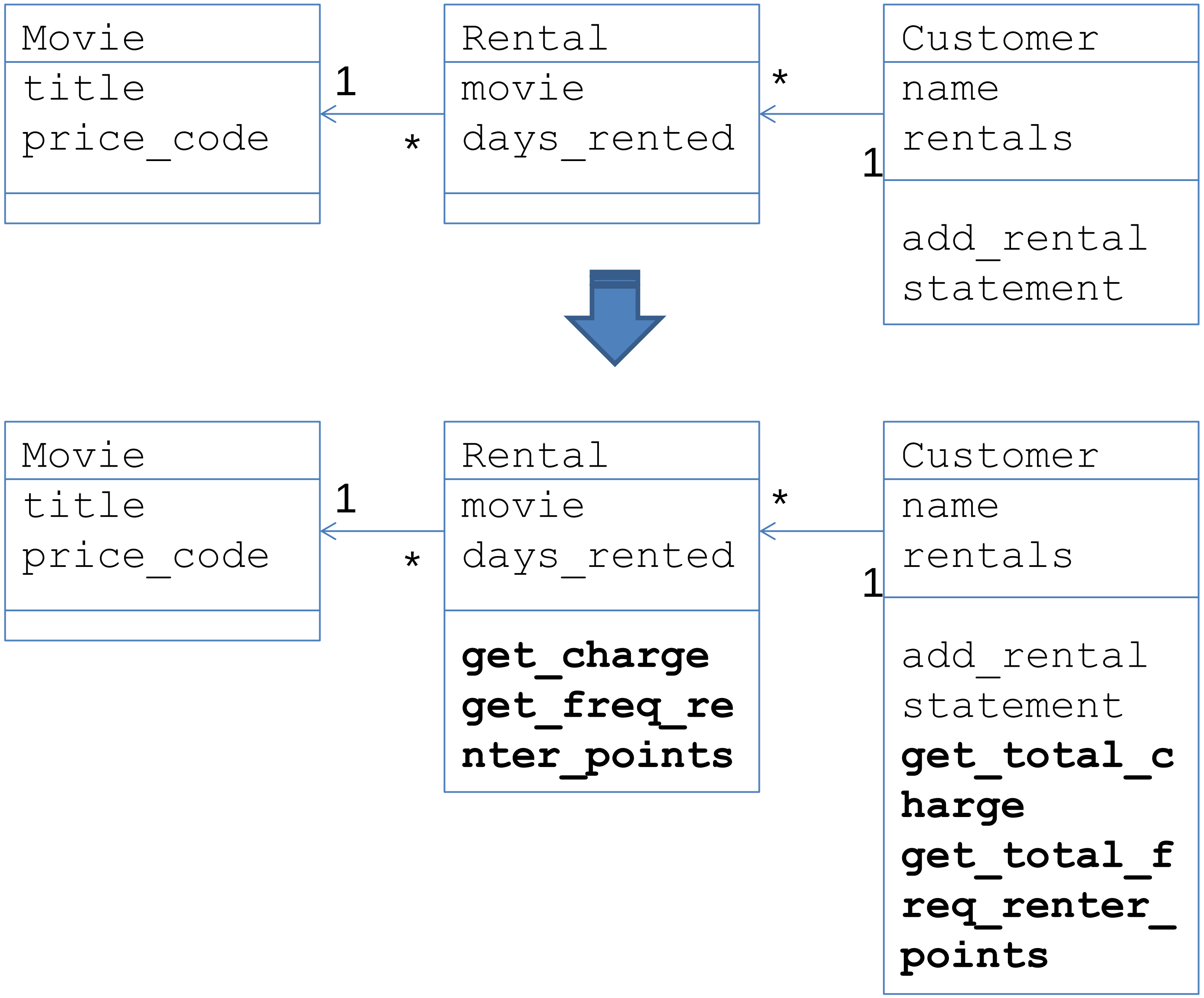
- http://jjhou.boolan.com/jjtbooks-refactoring.htm




Copyrighted Material
REFACTORING
IMPROVING THE DESIGN OF EXISTING CODE
MARTIN FOWLER
Contributions by Kent Beck, John Brant, William Opdyke, and Don Roberts
Foreword by Erich Gamma
Object Technology International Inc.
BOOCH JACOBSON RUMBAUGH
Copyrighted Material

- The file **‘dvdlib.py’** located in **lab/exercises/exercise15** is a replication of the sample program in the chapter 1 of the book ‘Refactoring’.
- We’re refactoring the `statement` method of the `Customer` class according the process of the “Decomposing and Redistributing the Statement Method” session in “Refactoring”.
- We’re using `unittest` to ensure that our each refactoring doesn’t break anything.

What Should You See



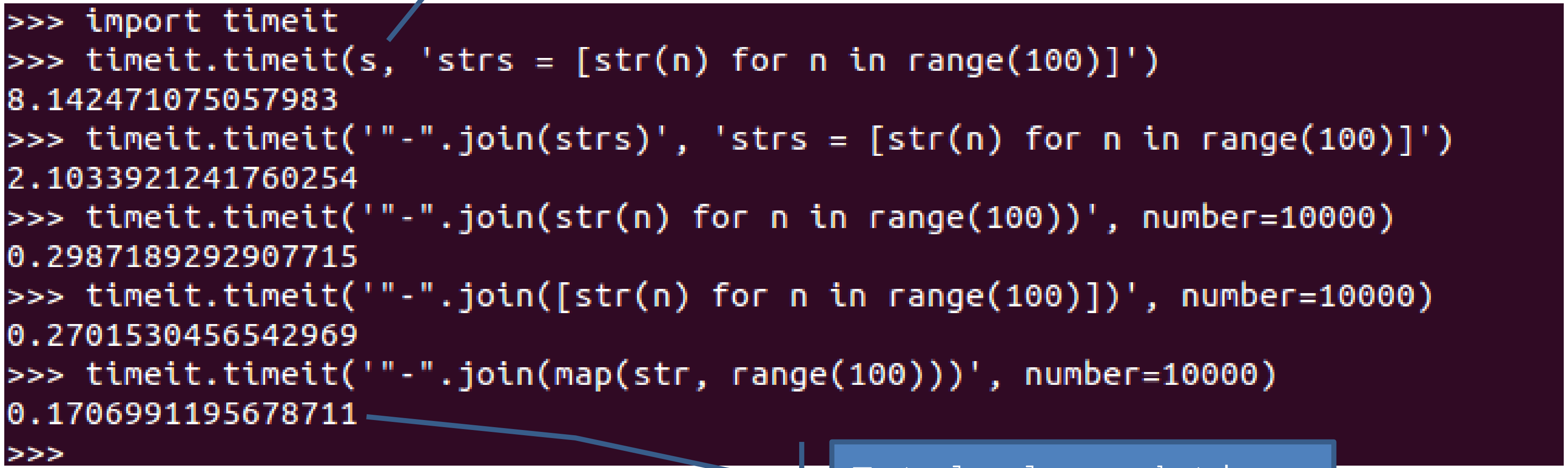
Profiling

- `timeit`
  - Measures execution time of small code snippets.
- `cProfile`
  - Describes the run time performance of a program.
  - Provides a variety of statistics.
  - Recommended for most users; it’s a C extension.
- `profile`
  - A pure Python module whose interface is imitated by `cProfile`, so they are mostly interchangeable; `cProfile` has a much lower overhead but is newer and might not be available on all systems.

timeit

```
s = '''\
all = ''
for s in strs:
    all += s
'''
```

- Python interface



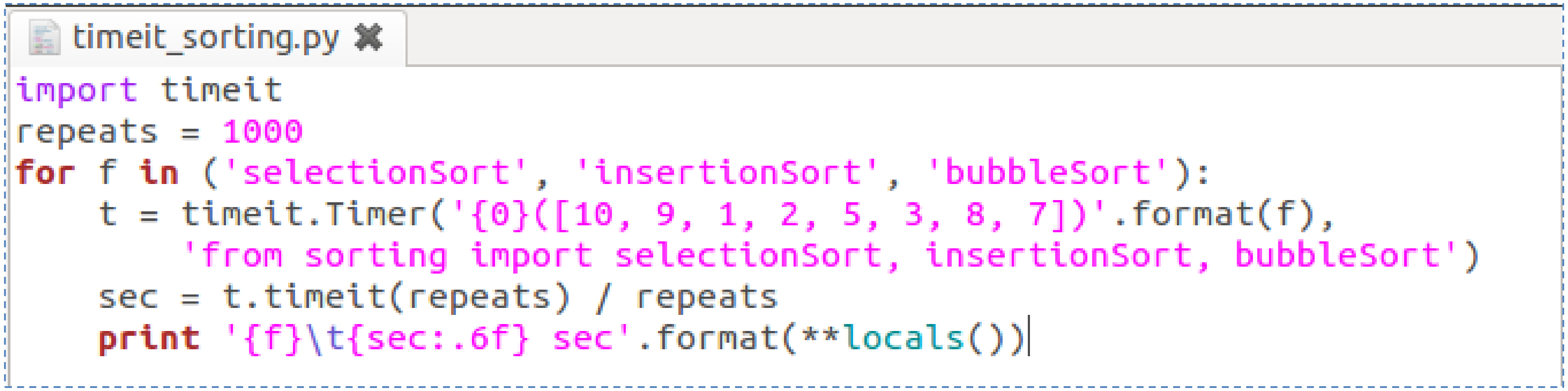
```
>>> import timeit
>>> timeit.timeit(s, 'strs = [str(n) for n in range(100)]')
8.142471075057983
>>> timeit.timeit('"-".join(strs)', 'strs = [str(n) for n in range(100)]')
2.1033921241760254
>>> timeit.timeit('"-".join(str(n) for n in range(100))', number=10000)
0.2987189292907715
>>> timeit.timeit('"-".join([str(n) for n in range(100)])', number=10000)
0.2701530456542969
>>> timeit.timeit('"-".join(map(str, range(100)))', number=10000)
0.1706991195678711
>>>
```

```
Total elapsed time,
in seconds.
```

- Command-Line Interface

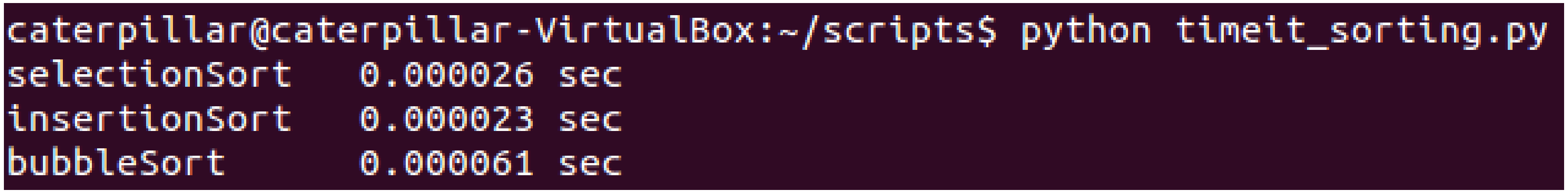
```
~$ python -m timeit '"-".join(str(n) for n in range(100))'
10000 loops, best of 3: 24.3 usec per loop
~$ python -m timeit '"-".join([str(n) for n in range(100)])'
10000 loops, best of 3: 21.9 usec per loop
~$ python -m timeit '"-".join(map(str, range(100)))'
100000 loops, best of 3: 16.7 usec per loop
```

A More Realistic Example



timeit_sorting.py

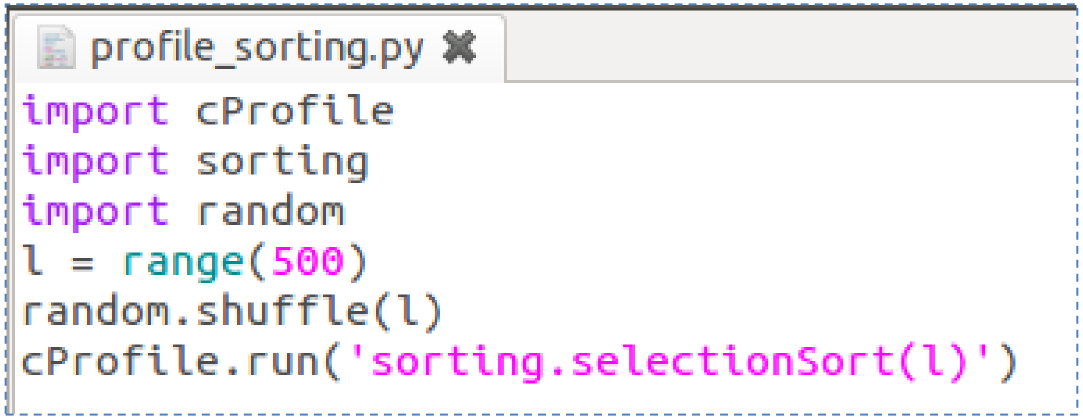
```
import timeit
repeats = 1000
for f in ('selectionSort', 'insertionSort', 'bubbleSort'):
    t = timeit.Timer('{0}([10, 9, 1, 2, 5, 3, 8, 7])'.format(f),
        'from sorting import selectionSort, insertionSort, bubbleSort')
    sec = t.timeit(repeats) / repeats
    print '{f}\t{sec:.6f} sec'.format(**locals())
```



```
caterpillar@caterpillar-VirtualBox:~/scripts$ python timeit_sorting.py
selectionSort   0.000026 sec
insertionSort   0.000023 sec
bubbleSort      0.000061 sec
```

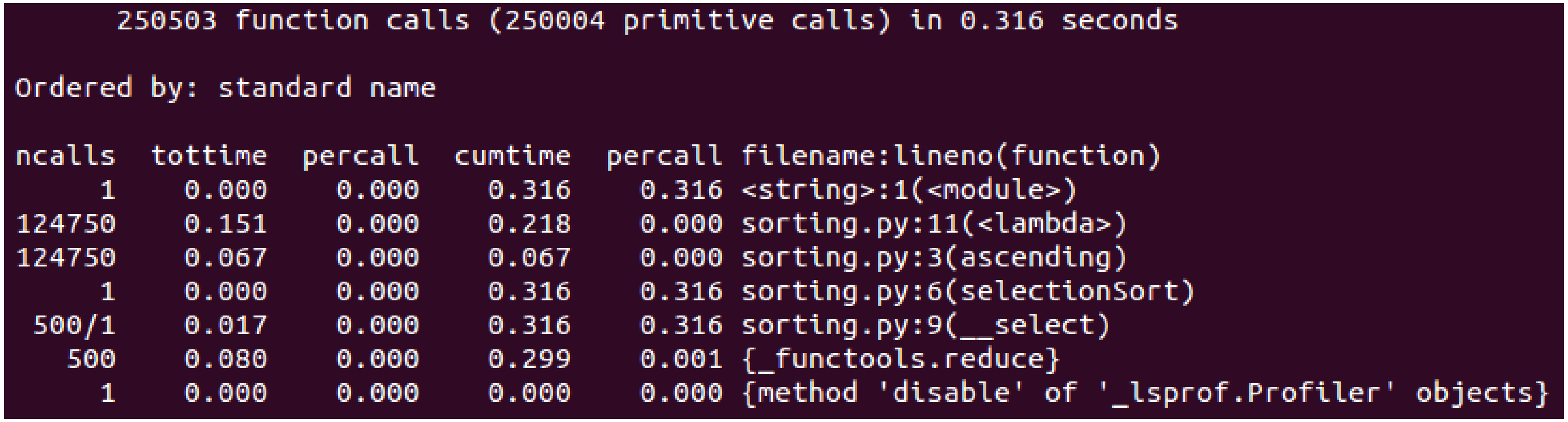
cProfile (profile)

- Profile an application with a main entry point



profile_sorting.py

```
import cProfile
import sorting
import random
l = range(500)
random.shuffle(l)
cProfile.run('sorting.selectionSort(l)')
```



```
         250503 function calls (250004 primitive calls) in 0.316 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    0.316    0.316 <string>:1(<module>)
   124750    0.151    0.000    0.218    0.000 sorting.py:11(<lambda>)
   124750    0.067    0.000    0.067    0.000 sorting.py:3(ascending)
        1    0.000    0.000    0.316    0.316 sorting.py:6(selectionSort)
    500/1    0.017    0.000    0.316    0.316 sorting.py:9(__select)
      500    0.080    0.000    0.299    0.001 {_functools.reduce}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
```

The Column Headings

- ncalls
  - “number of calls”, lists the number of calls to the specified function.
- tottime
  - “total time”, spent in the given function (and excluding time made in calls to sub-functions).
- percall
  - tottime / ncalls
- cumtime
  - “cumulative time”, spent in this and all subfunctions (from invocation till exit).
- percall
  - the quotient of cumtime divided by primitive calls.
- filename:lineno(function)
  - provides the respective data of each function

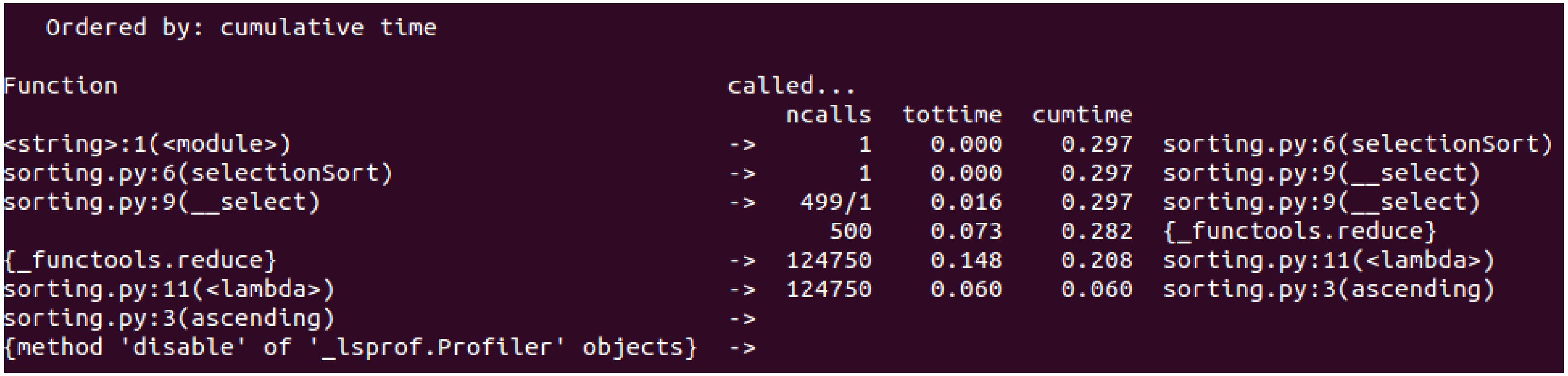
pstats

- To save the results of a profile into a file:

```
cProfile.run('sorting.selectionSort(l)', 'select_stats')
```

- To load the statistics data:

```
import pstats
p = pstats.Stats('select_stats')
p.strip_dirs().sort_stats('name').print_stats()
p.sort_stats('cumulative').print_stats(10)
p.sort_stats('time').print_stats(10)
```



```
   Ordered by: cumulative time

Function                                          called...
                                                      ncalls  tottime  cumtime
<string>:1(<module>)                              ->       1    0.000    0.297  sorting.py:6(selectionSort)
sorting.py:6(selectionSort)                       ->       1    0.000    0.297  sorting.py:9(__select)
sorting.py:9(__select)                            ->   499/1    0.016    0.297  sorting.py:9(__select)
                                                         500    0.073    0.282  {_functools.reduce}
{_functools.reduce}                               ->  124750    0.148    0.208  sorting.py:11(<lambda>)
sorting.py:11(<lambda>)                           ->  124750    0.060    0.060  sorting.py:3(ascending)
sorting.py:3(ascending)                           ->
{method 'disable' of '_lsprof.Profiler' objects}  ->
```

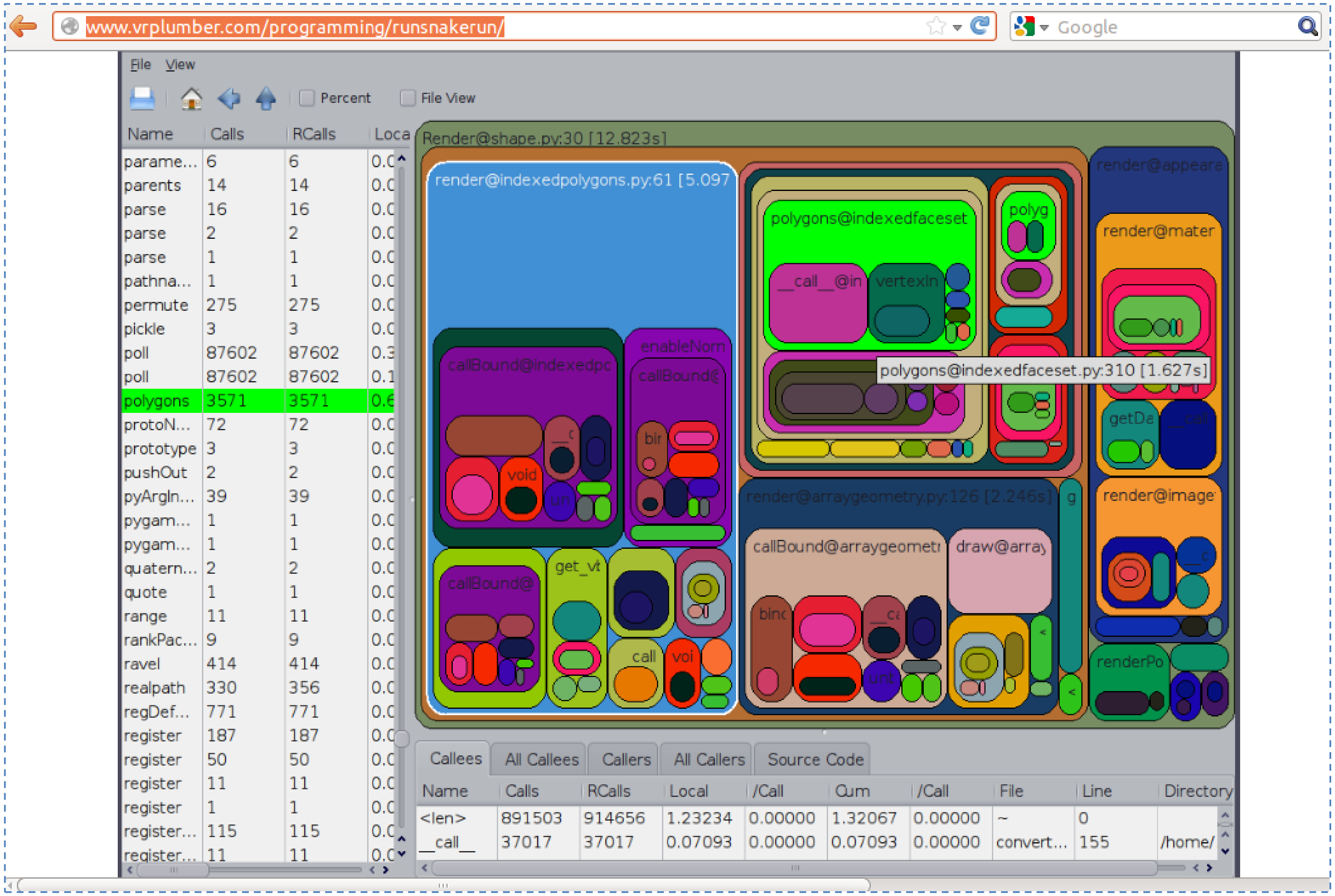
- The file cProfile.py can also be invoked as a script to profile another script.

```
python -m cProfile myscript.py
```

```
cProfile.py [-o output_file] [-s sort_order]
```

A Small GUI Utility

- http://www.vrplumber.com/programming/runsnakerun/



Name	Calls	RCalls	Loca
parame...	6	6	0.0
parents	14	14	0.0
parse	16	16	0.0
parse	2	2	0.0
parse	1	1	0.0
pathna...	1	1	0.0
permute	275	275	0.0
pickle	3	3	0.0
poll	87602	87602	0.3
poll	87602	87602	0.1
polygons	3571	3571	0.6
protoN...	72	72	0.0
prototype	3	3	0.0
pushOut	2	2	0.0
pyArgIn...	39	39	0.0
pygam...	1	1	0.0
pygam...	1	1	0.0
quatern...	2	2	0.0
quote	1	1	0.0
range	11	11	0.0
rankPac...	9	9	0.0
ravel	414	414	0.0
realpath	330	356	0.0
regDef...	771	771	0.0
register	187	187	0.0
register	50	50	0.0
register	11	11	0.0
register	1	1	0.0
register...	115	115	0.0
register...	11	11	0.0

Name	Calls	RCalls	Local	/Call	Cum	/Call	File	Line	Directory
<len>	891503	914656	1.23234	0.00000	1.32067	0.00000	~	0	
__call__	37017	37017	0.07093	0.00000	0.07093	0.00000	convert...	155	/home/

PyCon Taiwan

- PyCon Taiwan 2012
  - http://tw.pycon.org/2012/program/
- PyCon Taiwan 2013
  - http://tw.pycon.org/2013/en/program/

PyCon Taiwan 2012

- Even Faster Django
- Pyjamas - A Python-based Web Application Development Framework
- Developing Python Apps on Windows Azure
- PyKinect: Body Iteration Application Development Using Python
- STAF 在自動化測試上的延伸應用 – TMSTAF
- Qt Quick GUI Programming with PySide
- 所見非所得 - Metaclass 能幹嗎?

PyCon Taiwan 2013

- Use Pyramid Like a Pro
- MoSQL: More than SQL, and less than ORM
- 如何用 Django 在 24 小時內作出 prototype 微創業，以 petneed.me 為例
- Python memory management & Impact to memory-hungry application (DT)
- Dive into Python Class
- Python Coding Style Guide - 哥寫的 Python 是 Google Style
- Extend your legacy application with Python
- CPython 程式碼解析

- Extend your legacy application with Python
- CPython 程式碼解析
- 駭客看 Django
- 做遊戲學 Python
- Big Data Analysis in Python
- 周蟒 WEB 積木版與 Blockly
- The Life of an Ubuntu Developer
- 用 VPython 學 Python
- 當 Python 遇上魔術方塊

References

- Testing
  - http://docs.python.org/2/library/unittest.html
  - https://python-guide.readthedocs.org/en/latest/writing/tests/
- Profiling
  - http://docs.python.org/2/library/timeit.html
  - http://docs.python.org/2/library/profile.html
  - http://www.vrplumber.com/programming/runsnakerun/
- PyCon Taiwan
  - http://tw.pycon.org/2012/program/
  - http://tw.pycon.org/2013/en/program/