STRONG STEAM

Beginning Python

Ian Ozsvald

Kyran Dale

Emily Toop

Our background

- Python programmers for 10+ years
- Founders of video-based Python training site ShowMeDo.com
- Authors of several books
- Teacher at PyCon and EuroPython
- In StartupChile with computer vision startup StrongSteam.com

Goals

- Get you comfy with the command line and short Python files
- Writing, debugging, installing modules, solving tasks, finding help, writing reliable code
- The Python Challenges will make you think

Python history

- Python is a highly regarded, very reliable and very easy to learn programming language
- Created by Guido van Rossum (BDFL)
- Designed to be easy to learn – 'executable pseudocode'
- Started in 1989, Python 2.0 in 2000, Python 3.0 in 2008

About Python

- Procedural, Object Oriented
- Some functional components
- Automatic garbage collection
- Dynamic typing (any static typers here?)
- Strong typing (any weak typers here?)
- Case sensitive
- Late binding (dynamic name resolution)

Python environments

- CPython is 'normal' (python.exe)
- PyPy is new and cool
- Also Jython, IronPython, TinyPy, SL4A
- Cython and Shedskin for C compilation
- Shells – IPython (BPython) + WinPDB
- Editors – WingWare, PyDev, VIM/EMACS, TextPad (google “python editors”)

“Batteries Included”

- http://docs.python.org/library/
- numpy, NLTK, pyOpenCV
- Django, bottle, redis/mongodb, sqlite3
- win32COM, wxPython, pyGame, PIL
- SeleniumRC, webbrowser
- Templating, ORMs, Queues, Remote objs
- Inside: Blender, ArcGIS, OpenOffice

And you?

- Why do you want to use Python?

The Python Shell

- Start python.exe
- Python version?
- `exit()`
- `Ctrl d # Ctrl z`

(Some) immutable types

- `n = 42`
- `print n`
- `s = “the answer” # print s`
- `f = 22.3 # print f`
- `c = 2+3j # print c`
- What is a mutable type?
- http://docs.python.org/library/datatypes.html

Type checking

- `type(n)`
- `type(s)`
- `isinstance(n, int) # integer?`
- `isinstance(s, str) # string?`
- Q: What about f's type?

Making simple objects (again)

- `int(42)`
- `str(“the answer”)`
- `str(42)`
- `complex(2, 3)`
- Q: How to make a floating point object?

Comparisons

- `n == n`
- `n < 50`
- `n < 10`
- `n >= 0`
- `n == 42`
- `n == “42”`
- Q: What about f?

Simple conversions

- `str(n)`
- `int(str(n))`
- Q: Can you convert integer n to a floating point number?

None (like NULL)

- Has everyone come across None/NULL/null/Nothing/nil/0?
- `None`
- Q: What type is None?
- `x = 99`
- `x = None`
- Q: What type is x now?
- Q: How to ask if x is None?

Containers – list (mutable)

- `l = list()`
- `l = [] # a shortcut`
- `l = [1,2,3]`
- `len(l)`
- `l.append(4)`
- `len(l)`
- Q: How to ask “is the list length under 10”?

Containers - list

- `l = [1,3,2,5,0]`
- `l.sort() # did you get a result?`
- `l + [“something”]`
- `2 in l`
- Q: Is 99 in l?
- `l[:3] # slice up to`
- `l[2:] # slice from`

Containers – tuple (immutable)

- A list that can't be changed
- `t = (1,2)`
- `t = (1,2,”hi”)`
- `t[0]`
- Q: t[0] = 42
- Useful when we don't want things to change (e.g. dictionary keys)

Containers – set (mutable)

- `s = set()`
- `s = {1,2,3}`
- `s.add(4)`
- `s.add(4)`
- Q: Is 4 in s?
- Q: Is “hello” in s?
- Q: Length of s?

Containers - set

- `s2 = set([4,5])`
- `s.union(s2)`
- `s.intersection(s2)`
- `s.difference(s2) # symmetric_difference?`
- Q: Is s == s?
- Q: Is s == s2?
- Q: Length of union of s and s2?

Containers - Dictionaries

- `d = dict()`
- `d = {}`
- `d['ian'] = 35 # key, value`
- `d.items()`
- Q: Is 'ian' in d?

Containers – Dictionaries

- `d.keys()`
- `d.values()`
- `d.get("ian")`
- Q: d["someone"]
- Q: d.get on "someone"?

Speed and container choices

- Algorithm choice is critical to fit speed and memory constraints (but maybe not right now)
- Containers have different algorithms
- `l = range(10000000) # 10 mil.`
- `Q: 9999999 in l?`
- `s = set(l)`
- `Q: 9999999 in s?`

A quick test

- Q: We want to store more than 1 of the same thing so we choose...
- Q: We've got values assigned to keys...
- Q: We're after quick membership tests...

Peeking inside

- `dir()`
- `dir(l) # l = [1,2,3]`
- Q: What's inside s? # s = set(l)
- `help(l)`
- Q: Can we get help on l.append?
- Q: Can we get help on 42?

Modules

- "Batteries included"
- math
- string
- urllib
- json
- http://docs.python.org/library/

math module

- `import math`
- **don't** `from module import *`
- Modules are good, we shouldn't pollute global namespace
- `dir(math)`
- `help(math)`
- `math.pow(2,3) # and a shortcut`
- `math.sqrt(9)`

PythonChallenge 0

- http://www.pythonchallenge.com/

Writing a Python file

- Create `hello.py`
- `print “Hello”`
- `python hello.py`
- `name = raw_input(“Enter name:”)`
- `print “Hello {name}”.format(name=name)`
- `if __name__ == “__main__”:`

Looping (iteration)

- `for x in [1,2,3]:`
- `for x in “hello”:`
- `for x in range(10):`
- `for x in s: # s = set([1,2,3])`
- `for x in d: # d = {'ian':35,'bob':22}`
- `for x in d.items():`

Looping

```
n = 0
while n < 5:
    print n
    n = n + 1 # n += 1
    #break
```

Conditionals

```
n = 3
if n == 3:
    print “three”
else:
    print “other”
```

PythonChallenge 1 v1

- http://www.pythonchallenge.com/pc/def/map
- `text = "..."`
- Hints:
  - `for c in text`
  - `if c != " " # not equals`
  - `check if c is alpha`
  - `ord(c) + 2`
  - `ord('z')?`

PythonChallenge 1 v2

- `import string`
- `string.maketrans? # ipython`
- Q: Can we get a list of lowercase letters?
- Q: How do we get 3rd item onwards?
- Q: How do we get the first two items?
- Q: How do we join these together?
- `string.translate ...`

PythonChallenge 2 v1

- http://www.pythonchallenge.com/pc/def/ocr.
- Put the text in `soln2.py`
- Q: How do we print each character?
- Q: How do we test if c is an ascii_letter?
- Q: How do we build a string result?

PythonChallenge 2 v2

- Check Wikipedia for Regular Expression
- Q: Which pattern matches lowercase letters?
- `import re`
- `re.findall(pattern, “som3th1ng”)`
- Can one line get all the matching characters from `text`?

PythonChallenge 3 (optional)

- http://www.pythonchallenge.com/pc/def/equa
- We're after: `aAAAaAAAa`
- We want the middle letter
- Can we design a regular expression for this?
- Test regular expression on simple string
- Use `patt(e)rn` to extract our bit

Memory and Garbage Collection

- Automatic allocation and deallocation
- No new/delete or malloc/free
- No use of pointers
- Variables are references bound to objects
- id(obj) gives us the address (useful for debugging)

id 1

- `n = 1 # type(n)`
- `id(n)`
- `n = 2`
- `id(n)`
- `m = n; n += 1`
- Q: `id(m)`

id 2

- `s = “ian” # type(s)`
- `id(s)`
- `s2 = “ian”`
- Q: `id(s2)`?
- `s = “bob”`
- Q: `id(s)`?
- Q: `id(“ian”)`?

id 3

- Python shares references to immutable objects
- Try `l = []; m = []; id(l); id(m)`
- ints are pre-allocated and cached (saves time) # what does this mean for id?
- id won't change if a reference is kept
- If no references then GC kicks in

id 4

- `l = []`
- `l.append(2)`
- Q: What is `2`?
- Q: What is `l`?
- Q: Which is mutable?

id 5

- `l2 = [l]`
- `id(l2); id(l)`
- `l.append(42)`
- Q: What's in `l2` now?
- `l2[0].append(99)`
- Q: What's in `l`?
- Q: What's in `l2`?

id 6

- `l3 = [l, l]`
- `l.append(100)`
- Q: What's in `l3`?
- `l3[0].append(200)`
- Q: What's in `l`? `l3`?

Functions 1

```
def fn_name():
  “““a docstring”””
  #somework
  some_other_fn(...)
  return value # optional
answer = fn_name() # call it
```

- None is returned if 'return' isn't specified

Functions 2

```
def fn_name(arg1, arg2):
  “““a docstring”””
  #somework on arg1, arg2
  some_other_fn(...)
fn_name(arg1, arg2) # call it
```

- We don't have to receive an answer

Functions 3

```
def fn_name(optional_arg=42):
  “““a docstring”””
  #somework on optional_arg
fn_name() # call with default
fn_name(100) # call it
```

Functions 4

```
def fn_name(...):
  “““a docstring”””
  #somework
  return a, b, c # any number
(a, b, c) = fn_name(...)
```

- C only allows 1 returned item
- Python allows any number

Functions 5

- `def multiply` in `multiply.py`
- What shall we call the argument?
- What will the function do?
- Use a local variable, return the result
- How do we call it?
- Add a comment with `#` to explain the 1 line of work

Back to Functions

- `help(multiply)`
- Are we missing doc strings?
- `if __name___== “__main__”:`

Functions and locals

```
def fn_n(n_local):
  n_local += 10
n = 5; fn_n(n)
```

- Q: Has `n` been changed?
- Try it for a string too (`fn_s, s_local`)

Functions and locals

```
def fn_l(l_local):
  l_local.append(99)
l = []; fn_l(l)
```

- Q: Has `l` been changed?
- Try it for a set too (`fn_set, set_local`)
- What's different between the two cases?

Installing modules

- Avoid `easy_install` # old news
- Get pip `# get-pip.py`
- `python get-pip.py`
- `-> pip install pep8`

PEP8

- What is a PEP?
- Google for PEP8, have a read
- Do you use coding standards?
- `pep8 messy.py`
- Try it on your own code

pyLint

- Anyone used lint on C code?
- `pylint messy.py`
- Bad arguments?
- Missing docstrings?
- Bad imports?
- Try it on your own code

Python 3

- Python 2 is dead, 2.8 will never exist
- Python 3 is a bit different
- Thankfully `2to3` takes care of this
- Unit tests will keep you sane
- Changes: `print` now a function(), `unicode` by default, some things deprecated, some modules cleaned or moved, divide no longer truncates

Python 3

- `2to3 multiply.py # diff to stdout`
- What changed?
- `copy multiply.py multiply3.py`
- `2to3 -w multiply3.py # does it run in py2?`
- `3to2` exists online

Writing code – a quick review

- Prototype in the shell first
- Make small files, see `edit` in IPython
- Keep related stuff in one module
- Use TDD, always write tests
- Promote from the shell to modules to classes
- Keep interfaces clean and documented
- pyLint, PEP8

Learning more

- http://docs.python.org/tutorial/
- http://diveintopython.org/
- Python Tutor mailing list
- http://showmedo.com/ # disclosure
- http://www.doughellmann.com/PyMOTW/
- http://www.checkio.org/
- `import this`
- `import antigravity`

Sketching solutions?

- We probably don't have time
- But if we do – shall we talk through your use cases and plan some solutions?