0-PythonMastery

December 9, 2014

1 Python Mastery

Dec 8-11, 2014

Taught by:

- Ian Stokes-Rees ijstokes@continuum.io
  - Twitter: @ijstokes
  - About.Me: `http://about.me/ijstokes`
  - LinkedIn: `http://linkedin.com/in/ijstokes`

Course Website: `http://j.mp/python-mastery-isr19`
GitHub Version: `https://github.com/ijstokes/python-mastery-isr19.git`

2 Python Versions

`MAJOR.MINOR.MICRO`

E.g. `2.7.8`

- Major version: significant change to language structure (grammar and reserved words), and implementation
- Minor version: attempt to be forwards and backwards compatible, but introduce new *Standard Library* modules or updates to existing modules.
  - forwards/backwrads compatibility is not guaranteed
  - always give 2 minor versions advance notice of language changes
- Micro version: only bug fixes – no language or API changes

3 Great Python Tools

- IPython
  - command line: `ipython`
  - web view: `ipython notebook`
- Spyder Python IDE
  - included with Anaconda*
- PyCharm Python IDE
  - free community edition
- PyDev Eclipse IDE plugin for Python development
- Sublime Text 3 cross-platform programmer's editor
- Git

4 Reading and Reference

- Idiomatic Python: http://python.net/~goodger/projects/pycon/2007/idiomatic/handout.html
- Python Style Guides:
  - PEP 8: http://legacy.python.org/dev/peps/pep-0008/
  - Google Python Style Guide:http://google-styleguide.googlecode.com/svn/trunk/pyguide.html
- Magic methods:
  - official specs: http://docs.python.org/reference/datamodel.html#special-method-names
  - great summary: http://www.rafekettler.com/magicmethods.html

5 Python Language Mental Model

- 30 reserved words (aka keywords)
  - http://docs.python.org/2.7/reference/lexical_analysis.html#keywords
  - *logic:* `and, not, or`
  - *namespaces:* `import, from, as, del, global`
  - *object creation:* `class, def, lambda`
  - *functions:* `return, print, yield`
  - *looping:* `while, for, break, continue`
  - *conditional:* `if, else, elif`
  - *exeptions:* `try, except, finally, raise`
  - *misc:* `pass, assert, with, exec, in, is`
- Interpreter starts up knowing (almost) *nothing* but language syntax

6 `__builtin__`

- The Python Language defines a special module called ***builtin*** that is part of the Standard Library
- It contains *functions*, *exceptions*, and *classes* that are very common:
  - 3 special entries: *None*, *True*, *False* (keywords in Python 3)
  - 10 core types
    - *int, long, float, bool, complex, str, list, dict, tuple, set*
  - 20 supporting types
    - *file, xrange, object, . . .*
  - 40 exceptions (upper camel case, mostly ending in *Error* or *Warning*)
  - 50 functions
    - Math: *abs min max pow round sum divmod*
    - Logic: *all any apply map filter reduce*
    - Check: *callable isinstance issubclass*
    - Convert: *bin chr hex cmp coerce oct ord unichr*
    - Introspect: *dir id vars locals globals hasattr getattr setattr delattr compile eval execfile intern hash repr*
    - File: *open*
    - Iterable: *len range zip iter next sorted*
    - Other: *format reload*
- Any reference lookup that doesn't find the reference in the *local* namespace (first) or the *global* (which means *module*) namespace (second) will check the `__builtin__` modules namespace (third)

- CPython automatically provides a reference to the `__builtin__` module in every *global* namespace but gives it the name `__builtins__`
  - under normal use, you never need to use this module reference
- If the *local* or *global* namespace has a reference that is found in `__builtin__` then the `__builtin__` reference will be masked

6.1 Python Interpreter At Startup

Think of a *virgin python interpreter* as an *empty box*. This is what your program will see when it starts running, and what you will see when you start an interactive interpreter session.

The interpreter only knows about:

- Core sytnax
- `__main__` module scope (`local` and `global` scope both refer to this)
- how to resolve references:
  - *local scope* (`locals()`)
  - *global scope* (`globals()`)
  - `__builtin__` Standard Library module
- Interpreter starts with usual module meta-data references:
  - `__name__`: name of the module (always `__main__` for initial interpreter instance)
  - `__package__`: name of the package this module is in (always `None` for initial interpreter instance)
  - `__doc__`: module *docstring* (always `None` for initial interpreter instance)
  - *CPython implementation detail:* CPython interpreter will automatically add a reference `__builtins__` to all *global* scopes

Everything else needs to "appear" via:

1. *assignment statements*: `REFERENCE = EXPRESSION`, e.g. `x = 3` or `y = sort(['foo', 'bar', 'zip'])`
2. *import statements*: imports modules or creates references to objects defined in modules
3. *class statements*: creates a class object and adds a reference in the *local* scope
4. *def statements*: creates a function object and adds a reference in the *local* scope

6.2 Quiz: Python Mental Model

1. How many reserved words (key words) are there in Python (2.7)?
2. How many can you name? (*Hint:* think of the categories: namespace, logic, function, object creation, looping, conditional, exceptions, misc)
3. What does the Python interpreter know about at startup?
4. What is special about the `builtin` module?
5. What is the breakdown of the `builtin` module's contents?
6. How many packages are in the Python Standard Library?
7. What are features of the Python Standard Library?
8. How does reference scoping work in Python?
9. What is Python's reference lookup priority?
10. How are objects scoped in Python?
11. What are the mechanisms to create objects in Python?
12. How do you delete an object in Python?
13. How does `import` work?

7 Python Standard Library

- Python Standard Library offers about 300 amazing modules:
  - included with every Python distribution (*"batteries included"*)
  - well documented
  - stable APIs (7-12 years)
  - great performance (highly optimized – implemented in C if necessary)
  - widely used (field tested: problems will surface fast)
  - code reviewed
  - great test coverage (~100%)
- A killer feature of Python
  - so why wouldn't you use them?
  - sing it from the mountain tops!
- Check the Standard Library **first**:
  - before writing it yourself
  - before looking in the Cheeseshop
- Python Tutorial provides a nice overview in the last few chapters
  - skimmable
  - includes examples

```
In [1]: # Wakari helper function to clear namespace:
        def _clear():
            for thing in globals().keys():
                if not thing.startswith("_"):
                    del globals()[thing]

In [3]: # Wakari helper function to list non-underscore attributes
        def _dir(*obj):
            return [a for a in sorted(globals().keys()) if not a.startswith('_')]

In []:
```