An introduction to python
mcnet computing school 2016

Chris Pollard

University of Glasgow, MCNet

2016 05 16

Outline

- Intro
- Design choices
- Syntax
- Built-in types
- Libraries
- Intro to NumPy

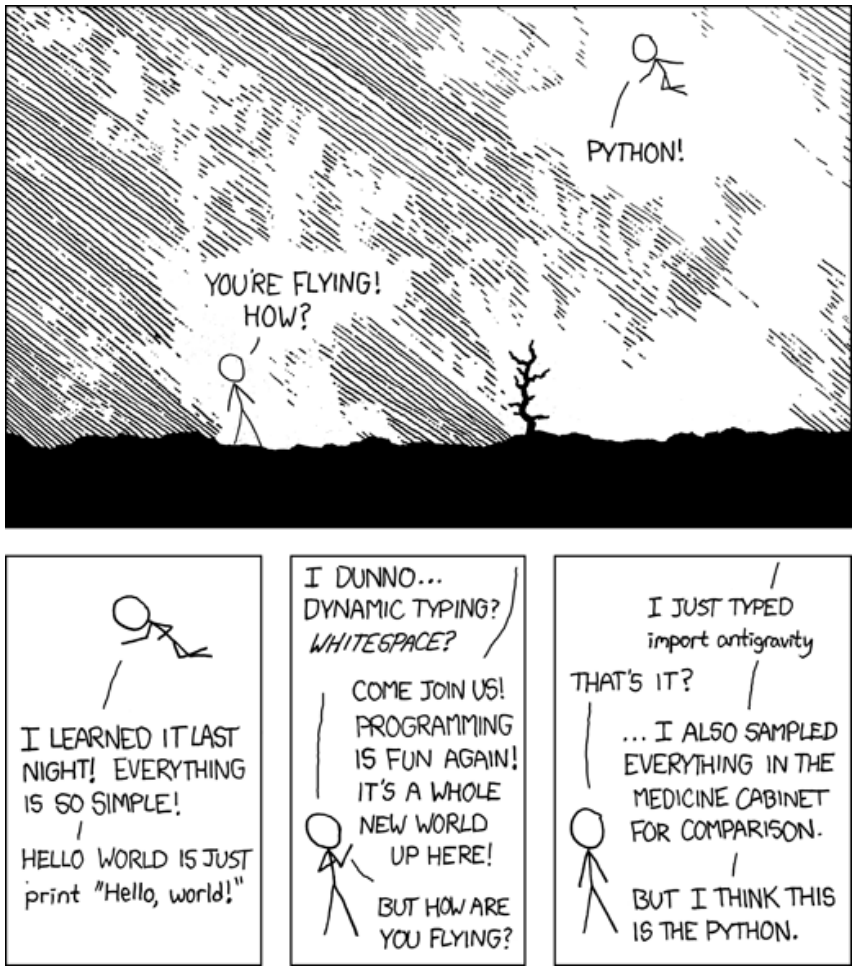
Python

- Python is a general purpose programming language (python.org)
- Development begun in 1989 by Guido van Rossum
- Available on many, many platforms (usually standard on UNIX)
- Latest versions: 2.7.11, 3.5.1
- Several implementations (usually CPython, but others for JVM, self-hosted, etc)

Why python?

- General purpose, high-level
- Code readability is a basic design ideal
- Supports many programming paradigms (object-oriented, imperative, functional, procedural, etc.)
- Widely supported with free + open reference implementation
- Large standard library and many third party libraries available

Why python?



It has a reputation for being beginner-friendly and fun to learn

Why not python?

So why wouldn't you use python for a project?

- Interpreted: if we never attempt to run a piece of code, then we don't know if it works.
- No compile-time errors (no type checking!)
- Speed
- More from the room?

There are ways to deal with each of these drawbacks, of course!

Zen of python

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
...
Readability counts.
...
Errors should never pass silently.
...
There should be one– and preferably only one –obvious way to do it.
...
If the implementation is hard to explain, it's a bad idea.
...

– Tim Peters

Which version should I use?

Personal recommendations:

- if you *know* you will run into compatibility issues (from e.g. collaboration), pick the version that won't cause technical problems.
- use the system installed version (unless there is no system version or it's very out-of-date, and you need newer features).
- use the latest version of python 3. most libraries now work with version 3.
- otherwise get the latest version of 2.7.

Before we dive in

From now on, fixed width code should be valid python.

If a line starts with ">>>", you can try it in python's REPL: just type "python" at you terminal (or install and run "ipython").

Most language keywords should be highlighted in blue.

The help() function is your friend!

Basic syntax

Python's syntax is quite similar to C/C++.

```
# the nth fibonacci number
def fib(n):
    if n < 2:
        return n
    else:
        return fib(n-1) + fib(n-2)
```

Notes:

- "def" starts a function definition.
- '#' starts a comment.
- indentation matters (tabs not recommended).
- no semicolons
- no braces

Control flows: for

```
# sum of first n fibonacci numbers
def fib_sum(n):
    s = 0
    # list of numbers from 0 to n.
    for i in range(n+1):
        s += fib(i)

    return s
```

Control flows: break, continue

```
# does something useless
def useless(n, m):
    s = 0
    for i in range(n, m):
        if i < 3:
            continue
        elif i > 27:
            break
        else:
            s += i

    return s
```

booleans

We can make things very clear even without comments:

```
def makeMeHappy(x, y, z):
    return isGood(x) \
           and not isBad(y) \
           and not isUgly(z):
```

Remember: “True” and “False” are capitalized in python.

import

Imports are simple:

```
# import the math library
import math
print(math.sqrt(10))
```

or

```
from math import sqrt
print(sqrt(10))
```

import

We can also import the math library with a new namespace name:

```
# import the math library
import math as m
print(m.sqrt(10))
```

This, however is highly discouraged due namespace clashes and difficulty in understanding what code is doing:

```
from math import *
print(sqrt(10))
```

But I need my braces

No.

```
>>> from __future__ import braces
  File "<stdin>", line 1
  SyntaxError: not a chance
```

One of many easter eggs in CPython...

Commonly used types

There are a bunch of handy, commonly-used types included in the standard library:

- Lists
- Tuples
- Strings
- Dictionaries (maps)
- Sets
- and more.

Lists and tuples

Main difference: lists are mutable; tuples are not.

```
>>> a = [1, 2, 3] # a list
>>> a[1] = 4
>>> a
[1, 4, 3]

>>> b = (1, 2, 3) # a tuple
>>> b[1] = 4
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  TypeError: 'tuple' object does not support \
    item assignment
```

Aside on mutability

Mutable objects (e.g. lists) have some important gotchas: for instance, in python they are always passed by *reference*, so we can easily (and accidentally) write code that does unexpected things.

```
>>> a = [1, 2, 3] # a list
>>> b = a
>>> b[1] = 4
>>> a
[1, 4, 3]

# this will change the second item in x to 5
# if x is mutable and fail otherwise!
def oops(x):
    x[1] = 5
```

Strings

```
>>> a = "hello!"
>>> a[1]
'e'

# this works but is slow---don't do in a loop!
>>> a += ' world!'
>>> a
'hello! world!'

# triple quotes: verbatim string
b = \
"""roses are red.
violets are blue.
Thankfully I ran out of space.
"""
```

String formatting

There are several ways to format strings; this is the preferred one:

```
>>> a = "hello {firstname} {lastname}!"
>>> a.format(firstname="Rick", lastname="Feyn")
'hello Rick Feyn!'
```

The preferred way to concatenate many strings is like so:

```
''.join(mystrings)

# join them with underscores
'_'.join(mystrings)
```

Dictionaries

Dictionaries are worth getting the hang of: they're very fast, heterogenous, mutable lookup tables.

```
>>> a = {"fact" : "Elvis lives",
         "fiction" : "Burger king kills"}
>>> a["fact"]
'Elvis lives'

>>> a["fiction"] = "for real"
>>> a[42] = "life"
>>> a
{42: 'life', 'fact': 'Elvis lives',
'fiction': 'for real'}
```

Type system

```
# typing is strong
>>> "hello" + 2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  TypeError: cannot concatenate 'str' and \
    'int' objects

# typing is dynamic
>>> a = 2
>>> a = "hello"

# we only check if x and y can be added
# when the + operator is called at runtime.
def addF(x, y):
    return x + y
```

The python standard library

. . . is, in short, huge.

- Regular expressions (re), text handling
- datetime, calendar
- numerical tools: math, decimal, fractions, random
- system calls, shell commands (os, sys, shutil, etc.)
- threads and multiprocessing
- pickle, sqlite3, zlib, bz2, tarfile, csv
- markup (xml, html), networking, graphics
- etc.

Third party packages

In addition to an extensive standard library, python has a very useful set of third party packages and distribution systems to handle them (e.g. pypi).

- Numerical python (numpy, scipy, matplotlib)
- Graphics (OpenGL)
- Computer vision (OpenCV)
- Database interfaces
- Web servers
- etc, etc, etc

Most packages can be easily installed with the pip command, however be careful: quality and maintenance can be a problem for pypi packages.

NumPy

- The NumPy/SciPy/matplotlib suite of external packages is worth familiarizing yourself with.
- It can improve the performance of your python code *a lot* by doing the heavy numerical calculations in compiled code...
- ...while providing a nice python API.

I'm going to go over some of the basic features and syntax based on the tutorial here.

NumPy arrays

The primary object added by NumPy is a homogenous, multidimensional array:

- table of elements *all of the same type*
- indexed by tuple of positive integers
- called "ndarray" or "array" (don't confuse NumPy arrays with python's built-in array.array object.)
- useful member variables: ndim, shape, size, dtype (use help() for more info!)

example

```
>>> import numpy as np
>>> a = np.arange(15).reshape(3, 5)
>>> a
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
>>> a.shape
(3, 5)
>>> a.ndim
2
>>> a.dtype.name
'int64'
>>> a.size
15
>>> type(a)
<type 'numpy.ndarray'>
```

array creation

```
>>> np.zeros( (3,4) )
array([[ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.]])
# dtype can also be specified
>>> np.ones( (2,3,4), dtype=np.int16 )
array([[[ 1, 1, 1, 1],
        [ 1, 1, 1, 1],
        [ 1, 1, 1, 1]],
       [[ 1, 1, 1, 1],
        [ 1, 1, 1, 1],
        [ 1, 1, 1, 1]]], dtype=int16)
```

array operations

Arithmetic functions are applied *elementwise*!

```
>>> a = np.array( [20,30,40,50] )
>>> b = np.arange( 4 )
>>> b
array([0, 1, 2, 3])
>>> c = a-b
>>> c
array([20, 29, 38, 47])
>>> b**2
array([0, 1, 4, 9])
>>> 10*np.sin(a)
array([ 9.12945251, -9.88031624,  7.4511316 , -2.
>>> a<35
array([ True, True, False, False], dtype=bool)
```

more array operations

"+=" and similar operators can be very efficient: operations are performed in place.

```
>>> a = np.ones((2,3), dtype=int)
>>> b = np.random.random((2,3))
>>> a *= 3
>>> a
array([[3, 3, 3],
       [3, 3, 3]])
>>> b += a
>>> b
array([[ 3.417022  ,  3.72032449,  3.00011437],
       [ 3.30233257,  3.14675589,  3.09233859]])
```

array folds

Additionally, a many common helper functions for folding over an array already exist:

```
>>> a = np.random.random((2,3))
>>> a
array([[ 0.18626021,  0.34556073,  0.39676747],
       [ 0.53881673,  0.41919451,  0.6852195 ]])
>>> a.sum()
2.5718191614547998
>>> a.min()
0.18626021137767091
>>> a.max()
0.68521950039675951
```

array slices

Remember: arrays are mutable. We can access items and set in an array similarly to how we do for list, but with a richer syntax.

```
>>> a = np.arange(5)**3
>>> a
array([0, 1, 8, 27, 64])
>>> a[2]
8
>>> a[2:]
array([8, 27, 64])
>>> a[0:4:2] = -1000
>>> a
array([-1000, 1, -1000, 27, 64])
```

multidimensional array slices

The array slicing syntax scales well into multidimensional arrays:

```
>>> b
array([[ 0,  1,  2,  3],
       [10, 11, 12, 13],
       [20, 21, 22, 23],
       [30, 31, 32, 33],
       [40, 41, 42, 43]])
>>> b[2,3]
23
# each row in the second column of b
>>> b[0:5, 1]
array([ 1, 11, 21, 31, 41])
```

but wait: there's more!

I highly recommend having a look at the NumPy/SciPy documentation to see what is available "for free" from these libraries: there is a lot!

Summary

That was a very quick intro to python and NumPy/SciPy.

There were many, many things not covered here (syntactic sugar, exceptions, classes and inheritance, advanced keywords, decorations, etc.)

The best way to learn it is to use it!

http://learnpythonthehardway.org/book/
http://projecteuler.net/
https://cryptopals.com/