# Python

2011-05-24

Bryce Boe

# Python Tidbits

- Python created by that guy --->
- Python is named after Monty Python's Flying Circus
- 1991 – Python 0.9.0 Released
- 2008 – Python 2.6 / 3.0rc2
- Current – Python 2.7.1 / 3.2
- *7th most popular language
  - Following: Java, C, C++, C#, PHP, Objective-C

Guido van Rossum

*http://www.tiobe.com/tiobe_index/index.htm

# Who / What Uses Python

- Google Appengine
- Youtube
- Yahoo! Groups
- Reddit
- Mercurial (Awesome VCS)
- Original BitTorrent client
- Dropbox (http://db.tt/EKo5BAw)
- Exaile (Audio player)
- MusicBrainz Picard

# Why Python

- High Level Language (HLL)
  - Rapid development
  - Very Readable Code
    - Up to 10x shorter code than Java/C/C++
  - Object Oriented
  - Dynamically Typed
- **Numerous** standard packages and modules
- **Numerous** third-party packages
  - Twisted, NumPy, SciPy, PIL, M2Crypto

# Standard Python Modules

- String Modules
  - string
  - re
  - StringIO / cStringIO
- Data Compression
  - gzip, bz2, zipfile, tarfile
- Data Structures
  - heapq, array, datetime, calendar
- Operating System
  - os, subprocess
- Networking
  - socket
  - urllib, urllib2, httplib, ftplib, poplib, imaplib, nntplib, smtplib, telnetlib
  - smtpd, SocketServer, BaseHTTPServer
- Others
  - crypt
  - hashlib
  - pickle / cPickle
  - threading

# Get on with it!

# Key Words

- and
- as
- assert
- break
- class
- continue
- def
- del
- elif
- else
- except
- exec
- finally
- for
- from
- global
- if
- import
- in
- is
- lambda
- not
- or
- pass
- print
- raise
- return
- try
- while
- with
- yield

# Types

- Immutable
  - Numbers
  - Strings
  - Tuples
- Mutable
  - Lists
  - Dictionaries
  - Sets
  - Most user defined objects

# Numbers

- The usual suspects
  - 12, 3.14, 0xFF, 0377, (-1+2)*3/4**5
  - abs(x), 0<x<=5
- C-style shifting & masking
  - 1 << 16, x & 0xff, x | 1, ~x, x^y
- Integer division truncates
  - 1/2 -> 0  # 1./2 -> 0.5, float(1)/2 -> 0.5
- Long (arbitrary precision), complex
  - 2**100 -> 1267650600228229401496703205376L
  - 1j**2 -> (-1+0j)

# Strings

- "hello"+"world" "helloworld" # concatenation
- "hello"*3 "hellohellohello" # repetition
- "hello"[0] "h" # indexing
- "hello"[-1] "o" # (from end)
- "hello"[1:4] "ell" # slicing
- "hello"[::2] "hlo" # more slicing
- len("hello") 5 # size
- "hello" < "jello” True # comparison
- "e" in "hello" True # search
- "escapes: \n etc, \033 etc, \if etc"
- 'single quotes' """triple quotes""" r"raw strings"

# More Strings

- Formatted Strings

  ```
  >>> "Pi: %.5f - Pi/2: %.5f" % (math.pi, math.pi/2)
  'Pi: 3.14159 - Pi/2: 1.57080'
  ```

- Splitting

  ```
  >>> "The quick brown fox jumps".split()
  ['The', 'quick', 'brown', 'fox', 'jumps']
  ```

- Joining

  ```
  >>> '?'.join(['The', 'quick', 'brown', 'fox', 'jumps'])
  'The?quick?brown?fox?jumps'
  ```

# Lists

- Flexible arrays, *not* linked lists
  - a = [99, "bottles of beer", ["on", "the", "wall"]]
- Same operators as for strings
  - a+b, a*3, a[0], a[-1], a[1:], len(a)
- Item and slice assignment
  - a[0] = 98
  - a[1:2] = ["bottles", "of", "beer"]
    -> [98, "bottles", "of", "beer", ["on", "the", "wall"]]
  - del a[-1]  # -> [98, "bottles", "of", "beer"]

# More List Operations

```
>>> a = range(5)                # [0,1,2,3,4]
>>> a.append(5)                 # [0,1,2,3,4,5]
>>> a.pop()                     # [0,1,2,3,4]
5
>>> a.insert(0, 42)             # [42,0,1,2,3,4]
>>> a.pop(0)            # [0,1,2,3,4]
42
>>> a.reverse()                 # [4,3,2,1,0]
>>> a.sort()                    # [0,1,2,3,4]
```

# Dictionaries

- Hash tables, "associative arrays"
  - d = {"duck": "eend", "water": "water"}
- Lookup:
  - d["duck"] -> "eend"
  - d["back"] # raises KeyError exception
- Insert, delete, overwrite:
  - d["back"] = "rug" # {"duck": "eend", "water": "water", "back": "rug"}
  - del d["water"] # {"duck": "eend", "back": "rug"}
  - d["duck"] = "duik" # {"duck": "duik", "back": "rug"}

# More Dictionary Operations

- Keys, values, items:
  - d.keys() -> ["duck", "back"]
  - d.values() -> ["duik", "rug"]
  - d.items() -> [("duck","duik"), ("back","rug")]
- Presence check:
  - d.has_key("duck") -> True
  - d.has_key("spam") -> False
  - "duck" in d -> True; "spam" in d -> False
- Values of any type; keys almost any
  - {"name":"Bryce", "age":22, ("hello","world"):1, 42:"yes", "flag": ["red","white","blue"]}

# Dictionary Details

- Keys must be **immutable**:
  - numbers, strings, tuples of immutables
    - these cannot be changed after creation
  - reason is *hashing* (fast lookup technique)
  - **not** lists or other dictionaries
    - these types of objects can be changed "in place"
  - no restrictions on values
- Keys will be listed in **arbitrary order**
  - again, because of hashing

# Tuples

- key = (lastname, firstname)
- point = x, y, z  # parentheses optional
- x, y, z = point  # unpack
- lastname = key[0]
- singleton = (1,)  # trailing comma!!!
- empty = ()  # parentheses!
- tuples vs. lists; tuples immutable

# Variables

- No need to declare
- Need to assign (initialize)
  - use of uninitialized variable raises exception
- Dynamically typed

```
if friendly: greeting = "hello world"
else: greeting = 12**2
print greeting
```

- ***Everything*** is a "variable":
  - Even functions, classes, modules

# Reference Semantics

- Assignment manipulates references
  - x = y **does not make a copy** of y
  - x = y makes x **reference** the object y references
- Very useful; but beware!
- Example:

```
>>> a = [1, 2, 3]
>>> b = a
>>> a.append(4)
>>> print b
[1, 2, 3, 4]
```

# Control Structure

if *condition*:
    *statements*
[elif *condition*:
    *statements*] ...
[else:
    *statements*]

while *condition*:
    *statements*
[else:
    *statements*]

for *var* in *sequence*:
    *statements*
[else:
    *statements*]

break
continue

# Grouping Indentation

## C

```
int i;
for (i = 0; i < 20; i++) {
    if (i%3 == 0) {
        printf("%d\n", i);
        if (i%5 == 0) {
            printf("Bingo!\n"
                   );
        }
    }
    printf("---\n");
}
```

```
0
Bingo!
---
---
---
3
---
---
---
6
---
---
---
9
---
---
---
12
---
---
---
15
Bingo!
---
---
---
18
---
---
```

## Python

```
for i in range(20):
    if i%3 == 0:
        print i
        if i%5 == 0:
            print "Bingo!"
    print "---"
```

# Functions, Procedures

```
def name(arg1, arg2=some_val, ..., *v, **kw):
    """documentation""" # optional doc string
    statements

return                  # from procedure
                        # implied None
return expression       # from function
```

# Example Function

```
def gcd(a, b):
    "greatest common divisor"
    while a != 0:
        a, b = b%a, a    # parallel assignment
    return b

>>> gcd.__doc__
'greatest common divisor'
>>> gcd(12, 20)
4
```

# Classes

class *name(object)*:

    "*documentation*"

    *statements*

-or-

class *name*(*base1*, *base2*, ...):

    *...*

Most, *statements* are method definitions:

    def *name*(self, *arg1*, *arg2*, ...):

        *...*

May also be *class variable* assignments

# Example Class

```
class Stack:
    "A well-known data structure…"
    def __init__(self):                 # constructor
        self.items = []
    def push(self, x):
        self.items.append(x)            # the sky is the limit
    def pop(self):
        x = self.items[-1] # what happens if it’s empty?
        del self.items[-1]
        return x
    def is_empty(self):
        return len(self.items) == 0         # Boolean result
```

# Using Classes

- To create an instance, simply call the class object:

```
x = Stack()        # no 'new' operator!
```

- To use methods of the instance, call using dot notation:

```
x.is_empty()       # -> True
x.push(1)          # [1]
x.is_empty()       # -> False
x.push("hello")    # [1, "hello"]
x.pop()            # -> "hello"     # [1]
```

- To inspect instance variables, use dot notation:

```
x.items            # -> [1]
```

# Subclassing

```
class FancyStack(Stack):
    "stack with added ability to inspect inferior stack items"

    def peek(self, n):
        """peek(0) returns top; peek(1) returns item below
    that; etc."""
        size = len(self.items)
        assert 0 <= n < size            # test precondition
        return self.items[size-1-n]
```

# Subclassing (2)

```
class LimitedStack(FancyStack):
    "fancy stack with limit on stack size"

    def __init__(self, limit):
        self.limit = limit
        FancyStack.__init__(self)      # base class constructor

    def push(self, x):
        assert len(self.items) < self.limit
        FancyStack.push(self, x)       # "super" method call
```

# Modules

- Collection of stuff in *foo*.py file
  - functions, classes, variables
- Importing modules:
  - import re; print re.match("[a-z]+", s)
  - from re import match; print match("[a-z]+", s)
- Import with rename:
  - import re as regex
  - from re import match as m

# Packages

- Collection of modules in directory
- Must have __init__.py file
- May contain subpackages
- Import syntax:
  - from P.Q.M import foo; print foo()
  - from P.Q import M; print M.foo()
  - import P.Q.M; print P.Q.M.foo()
  - import P.Q.M as M; print M.foo()  # new

# Catching Exceptions

```
def foo(x):
    return 1/x

def bar(x):
    try:
        print foo(x)
    except ZeroDivisionError, message:
        print "Can’t divide by zero:", message
    else:
        # Only executed if no exceptions are caught
        pass

bar(0)
```

# Try-finally: Cleanup

```
f = open(file)
try:
    process_file(f)
finally:
    f.close()          # always executed
print "OK" # executed on success only
```

or

```
with open(file) as f:
    process_file(f)
```

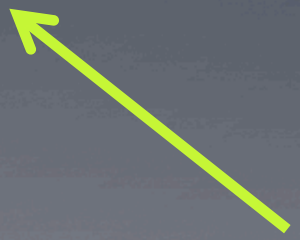
# File Objects

- f = open(*filename*[, *mode*[, *buffersize*])
  - mode can be "r", "w", "a" (like C stdio); default "r"
  - append "b" for text translation mode
  - append "+" for read/write open
  - buffersize: 0=unbuffered; 1=line-buffered; buffered
- methods:
  - read([*nbytes*]), readline(), readlines()
  - write(*string*), writelines(*list*)
  - seek(*pos*[, *how*]), tell()
  - flush(), close()
  - fileno()

# A Program in C

```
 1. #include <stdio.h>
 2. #include <stdlib.h>
 3. int main() {
 4.   srand(time());
 5.   int i, j, random_odd[10];
 6.   // Generate random odd integers
 7.   for (i = j = 0; i < 10; i++) {
 8.     random_odd[j] = rand() % 10000;
 9.     if (random_odd[j] % 2)
10.       j += 1;
11.   }
12.   for (i = 0; i < j; i++)
13.     printf("%d\n", random_odd[i]);
14.   // Calculate the product
15.   unsigned long product;
16.   for (i = 0; i < j; i++)
17.     product *= random_odd[i];
18.   printf("\n%lu\n", product);
19. }
```

```
lappy2:~ bryce$ ./a.out
4989
3881
9457
3049
843
6709
7813
7281

2062446592
```

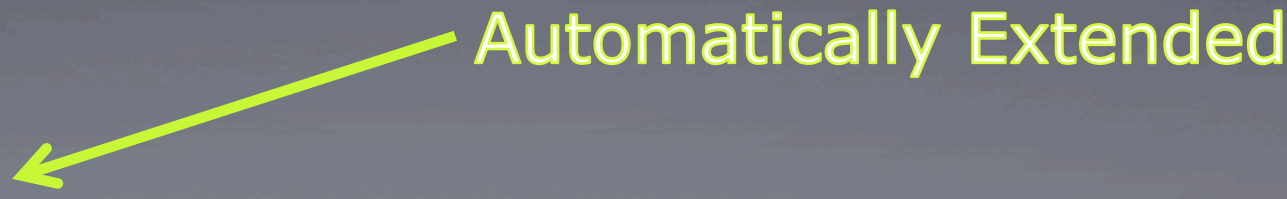


Overflowed

# And in Python

```
1. #!/usr/bin/env python
2. import random
3.
4. # Generate random odd integers
5. # Using a list comprehension
6. random_odd = [x for x in random.sample(range(10000), 10) if x % 2]
7. for num in random_odd:
8.     print num
9.
10. # Calculate the product
11. print '\n', reduce(lambda x, y: x*y, random_odd)
```

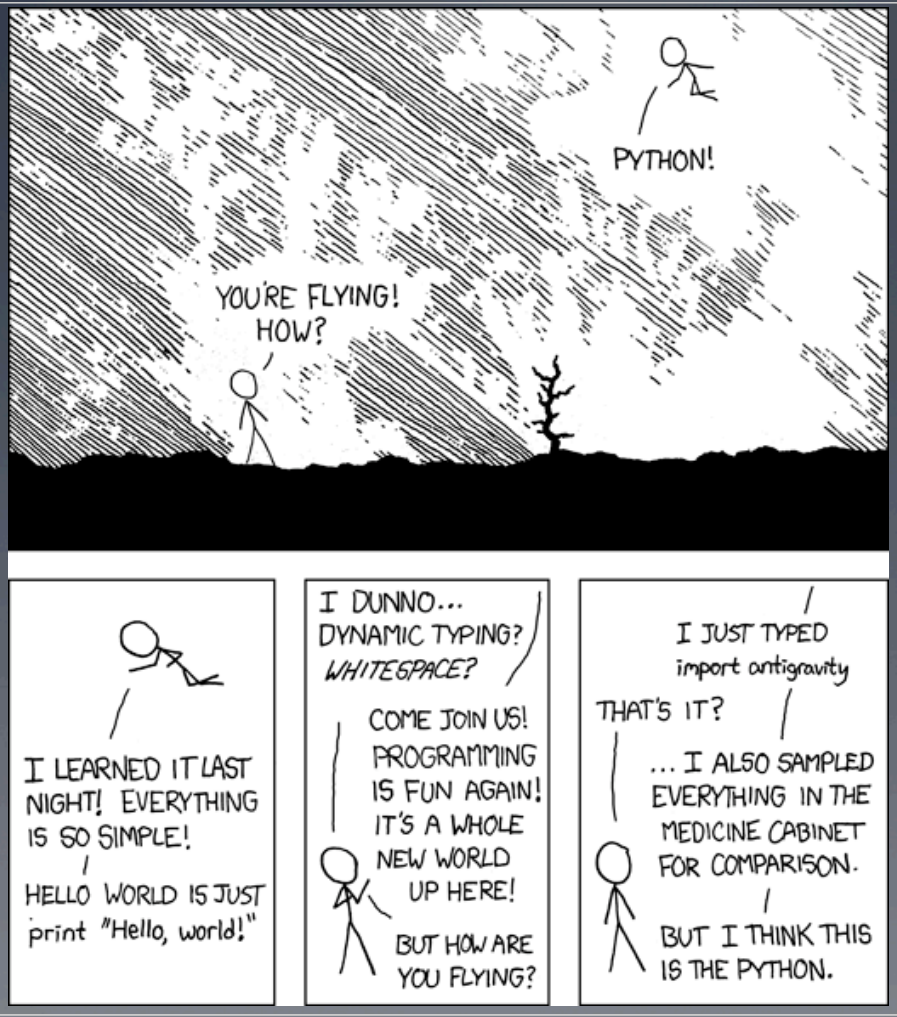
```
lappy2:~ bryce$ ./blah.py
1303
1987
4959
8389

107707658703111
```



# References

- Python Tutorial:
  - http://docs.python.org/tutorial/
- Python Documentation:
  - http://docs.python.org/library/index.html
- Many slides from:
  - http://www.python.org/doc/essays/ppt/lwnyc2002/intro22.ppt

# Questions?


PYTHON!
YOU'RE FLYING! HOW?
I LEARNED IT LAST NIGHT! EVERYTHING IS SO SIMPLE!
HELLO WORLD IS JUST print "Hello, world!"
I DUNNO... DYNAMIC TYPING? WHITESPACE?
COME JOIN US! PROGRAMMING IS FUN AGAIN! IT'S A WHOLE NEW WORLD UP HERE!
BUT HOW ARE YOU FLYING?
I JUST TYPED import antigravity
THAT'S IT?
... I ALSO SAMPLED EVERYTHING IN THE MEDICINE CABINET FOR COMPARISON.
BUT I THINK THIS IS THE PYTHON.