



An Introduction to Python and Python Web Programming

Gabe Rudy, Director of Software Development
Golden Helix Inc.



About Me

- Gabe Rudy
- Software developer at Golden Helix since 2002
- MSU CS alumnus Dec. 2005
- Responsibilities:
 - Team lead
 - Making “product vision” a reality
 - Scouting new technologies
 - Coding standards/best practice/architecture



Agenda

- Introduction
- **About Python**
- Data Types
- Control Flow
- Functions/Modules/Classes
- Web Programming
- Q&A and Resources



Python

- Dynamic typed
- Often compared to Tcl, Perl, Ruby, Scheme, Java
- Very clear, readable syntax
- Powerful introspection
- Modular, hierarchal packages
- Exception handling



Python Everywhere

- Embeddable (inside C++/Java/.Net applications)
- Byte-compiled
- Ported to different runtimes:
 - Jython – Python for JVM
 - IronPython – Python for CLR (.NET/Mono)
- Cross-platform
 - Preinstalled on Mac OS X and Linux
- GUI agnostic, but many options



Using Python

- Interactive (demo)
- From file – usually with .py suffix
- Script (Linux/Mac)

- myfile.py:

```
#!/usr/bin/python
print "Hello World\n"
```

- run like a program

```
bash# chmod a+x myfile.py
bash# ./myfile.py
Hello World
bash#
```



Agenda

- Introduction
- About Python
- **Data Types**
- Control Flow
- Functions/Modules/Classes
- Web Programming
- Q&A and Resources



Python Data Types

- **Numbers:** `10, 0.746, 123456789123456789L`
- **Strings:** `"Double", 'Single', """span lines
"use" 'quotes' etc"""`
- **Lists:** `[10.44, 1, 2, "string", [4, False]]`
- **Tuples:** `("immutable lists", 30, (2.22, None))`
- **Dictionaries:** `{'a': 10, 'b': 'foo', 123: ['bar', 1]}`
- **Objects:** `MyObject("constructor arguments", 30)`
- **Modules:** `import myfile, sys`



Numbers

- **Hex:** `0x0f`, `0xFFFF`, `0x0`, `0x123456789ABCDEF`
- **Floats:** `0.123e-10`, `10E30`, `0.0000000123`
- **Complex:** `c = 1.2+5.6j`; `c.real`; `c.imag`
- **Int:** `4/3 = 1`, `4.0/3 = 1.333`, `9%2 = 1`, `2**8 = 256`
- **Long:** Infinite precision, bounded by memory only, int is automatically converted to a long if it would overflow
 - `1000000 * 100000 = 100000000000000L`
- **Math:** `from math import *: pi, e, sin(), sqrt(), ceil(), pow(), fabs(), log()`



More on Operations

- **Standard:** `+, -, *, /, %(modules), ** (power)`
- **Bitwise:** `&, |, ^, ~, <<, >>`
- **Conversion:** `int(10.2), long(10), float(10), complex(10, 2)`
- **Rounding:** `round(1.23456789, 2) = 1.23`



Strings

- Immutable
- Bytes and null: "... \xFF\x23,\0\n\t\" too"
- Unicode: u"asdf", u"æé®ß"
- Functions: "a,b,c".split(",") = ["a","b","c"]
- Literal choices: "use'like'this"
- 'or"this"'
- '''triple"single quote'''
- """or use triple
double"quotes"""



String Functions and Formatting

- `"Python".upper() = "PYTHON"`
- `"Python".find("th") = 2`
- `replace(from,to), strip(), lstrip(), rstrip()`
- `str(10) = "10"`
- **Formating like C printf, syntax:** `format % args`
- `"%.2f %d"%(1.123,123) = "1.23 123"`
- `"str: %s "%('string') = "str: string "`
- `"hex: 0x%x" % 15 = "hex: 0xf"`



Lists and Tuples

- Can include any object (primitive) including lists
- Indexing: `a = [1,2,3]; a[1] => 2`
- Assignment: `a[2] = 4`
- Methods:
 - `append(x)` - add `x` to end
 - `extend(L)` - appends items in `L`
 - `insert(i,x)` - insert `x` at position `I`
 - `remove(x)` - remove first instance of `x`
 - `index(x)` - index of first instance of `x`
 - `count(x)` - num occurrences of `x`
 - `sort()`, `reverse()`
- Tuples are immutable lists (leaner): `(2, 'asdf', 1.2)`



Slicing Sequences

- Like indexing but gives you a range
- 'Go Bobcats'[3:6] => 'Bob'
- 'Bobcats'[3:] => 'cats'
- 'Bobcats'[:3] => 'Bob'
- 'Bobcats'[:] => 'Bobcats' (copied)
- 'Bobcats'[-1] => 's'
- 'Go Bobcats'[-4:] => 'cats'
- 'Go Bobcats'[:-4] => 'Go Bob'
- 'Bobcats'[100:200] => "" (convenient?)



Dictionaries

- Associative arrays
- Indexed by any hashable immutable type
- `d1 = {1:2, 'foo': ['bar', 3]}`
- `d1[1] => 2`
- `d1['foo'] => ['bar',3]`
- `d1.keys() => [1, 'foo']`
- `d1.values() => [2, ['bar'],3]`
- `d1.items() => [(1,2),('foo',['bar', 3])]`
- `d1[1] = 'overwrite value'`
- `d1[2] = 'new value'`



Agenda

- Introduction
- About Python
- Data Types
- **Control Flow**
- Functions/Modules/Classes
- Web Programming
- Q&A and Resources



Basics and Whitespace

- Whitespace is syntax and indicates blocking
- Think of it like pushing/popping a whitespace stack: must be consistent within scope
- Editors make it trivial (auto indenting etc)
- Statement continuations are not tied to whitespace rules (sequences and dict literals, triple quote strings, function arguments and other operations in parentheses)



Statements

- **Null statement:** `pass`
- **Delete:** `del var1, func1, etc`
- **Print:** `print "string with formating %d"%20`
 - `print var1`
 - **Can just type var1 when running interactively**
 - `print var1,var2 #comma adds implicit space`
 - `print var1, #trailing comma, no '\n'`
 - `print >> someFile, "some text" #use file.write()
instead`



Control Flow

```
if 10 in range(1,20):
    i = 10
    nums = []
    def plusOne(n):
        return n + 1
    while i < 20:
        if i == 15: continue
        elif i > 12: i = plusOne(i)
        else: nums.append(i)
    for val in nums:
        print val
        if val % 2 == 0 and val > 17:
            break
```



Exceptions/Assertions

```
assert expr[, message]
try:
    statements
    raise ValueError(message)
except Exception, e:
    do stuff with e
else:
    no exception raised
```



List comprehension

- `result = [expression for item in sequence]`
- `odds = [num for num in range(0,200) if num%2==1]`
- `evens = [2 * i for i in range(1,100)]`
- `together = [x+y for x,y in zip(odds,evens)]`
- `stuff = [[w.upper, len(w)] for w in dir()].join('\t')`



Agenda

- Introduction
- About Python
- Data Types
- Control Flow
- **Functions/Modules/Classes**
- Web Programming
- Q&A and Resources



Functions and Callables

- `def functionName(params): pass`
- **positional:** `def f1(first, second, *rest)`
- **keyword:** `def f2(first=10, second=None, **rest)`
- **keyword args optional and not ordered:**
 - `f2()`
 - `f2(second='second arg')`
 - `f2(other="caught by rest", first="not ordered")`
- **mix:** `def f3(first, second, option1=10, option2="")`
- **Can return tuple and “unpack” it to simulate multiple return values**

```
def returnManyThings():  
    return 10, "Bobcats", returnManyThings  
a,b,c = returnManyThings()
```



Modules

- Any python file can be imported as a module
- Search path: current directory, sys.path
- package is any directory with a `__init__.py` file in import path

```
import myfile
myfile.myFunc()
import myfile as MyUtils
MyUtils.myFunc()
from myfile import myFunc as superFunc
superFunc
from distutils.util import spawn
help(spawn)
from sys import *
print version
```



Objects (brief)

```
class MyFoo(object):
    def __init__(self, foo):
        self.foo = foo
    def getFoo(self):
        return self.foo
    def setFoo(self, newFoo):
        self.foo = newFoo
    def fooStuff(self, stuff):
        return self.foo * stuff

class SubBar(MyFoo):
    def __init__(self, bar):
        MyFoo.__init__(self, bar)

myBar = SubBar(10)
myBar.fooStuff(5) => 50
SubBar.fooStuff(myBar, 5) => 50
```



Objects (cont)

- `isinstance(val, type), issubclass(inst, classType)`
- **Subclass base types:** `class MyInt(int): pass`
- `__del__(self)` When object is deleted
- In OO sense, all functions are “virtual”
- Polymorphism works, but take one step further with “duck typing”



Agenda

- Introduction
- About Python
- Data Types
- Control Flow
- Functions/Modules/Classes
- **Web Programming**
- Q&A and Resources



Agenda

- Introduction
- About Python
- Data Types
- Control Flow
- Functions/Modules/Classes
- Web Programming
- **Q&A and Resources**



What's out there

- Lots of stuff
- DB-API – one interface to database connector modules (PostgreSQL, SQLite, MySQL, etc)
- Database abstraction layers to access databases like python objects
 - `john = Person(firstName="John",...)`
 - `Person.selectBy(firstName="John")`
- Can be optimized to scale (Google uses it)



CGI

- POST or GET
- I use it to handle some AJAX requests
- cgi module handles getting form data
- cgitb module prints nice traceback messages to HTML for when you mess up
- Secret Santa example/demo



Q&A and Resources

- Python.org
- Google “Python Quick Reference”
- web.py – Very simple web application framework
- Django – MVC and database friendly
- TurboGears – Powerfull, RoR like
- Twisted – Really extensive async networking
- Trac – powerfull bug tracking system