Simplistix

An Introduction to Python

Chris Withers

Who am I?

- Chris Withers
- Independent Zope and Python Consultant
- Using Python since 1999

- What do I use Python for?
  - Content Management
  - Systems Integration
  - XML manipulation

Simplistix

chris@simplistix.co.uk
http://www.simplistix.co.uk

“The Plan”

- Tour through python
- Stopping for questions
- “How do I do that?” session

chris@simplistix.co.uk
http://www.simplistix.co.uk

Pre-requisites

- Knowledge of at least one programming language
- Understanding of object oriented programming

chris@simplistix.co.uk
http://www.simplistix.co.uk

Why Python?

- Looks like pseudo code
- Quicker to develop
  - 5-10 times faster than C
  - 1-3 times faster than Java
- Faster than interpretted
  - While still being interpretted!
  - (not like Java!)
- Everything is an object
- Big “batteries included” library

chris@simplistix.co.uk
http://www.simplistix.co.uk

Executing Code

- Interactive shell

```
C:\>python
Python 2.2 (#28, Dec 21 2001, 12:21:22) [MSC 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more
   information.
>>> print "hello world"
hello world
>>>
```

- Running a script

```
C:\>python HelloWorld.py
hello world
C:\>
```


chris@simplistix.co.uk
http://www.simplistix.co.uk

An Example

- Telephone directory
  - Add
  - Remove
  - Lookup
  - Import
- Lets have a look…

chris@simplistix.co.uk
http://www.simplistix.co.uk

Code Layout

- Scope from indentation
  - this is a good thing!
- Small number of keywords
  - def, global, try, except, del, print, for, if, else, elif, is, in, class, import, from, while
- Comments
  - Start with a #
  - No block comments
- Docstrings

chris@simplistix.co.uk
http://www.simplistix.co.uk

The Basics

- Expressions
  - Result in a value

```
>>> 1.5*2
3.0
>>> (5-2)*3
9
```

- Variables
  - Dynamic typing

```
>>> x,y,z=2,3,4
>>> z = 2*x + y
>>> z
7
```

- Variable assignment is NOT an expression!

chris@simplistix.co.uk
http://www.simplistix.co.uk

Data Types – Numbers

- Integer
- Long
  - No limit
  - Implicit conversion from integer
- Floating point
- “correct” maths soon
  - 1 / 2 = ?
- Multiple range testing
  - w < x < y < z

chris@simplistix.co.uk
http://www.simplistix.co.uk

Data Types – Sequences

- They can…
  - Be iterated over
  - Indexed
- Tuple
  - immutable
- List
  - Muttable

chris@simplistix.co.uk
http://www.simplistix.co.uk

Data Types - Sequences

- Lots of handy manipulation

```
# create a list
L = range(5)

# add an element to the end of the list
L.append(-3)

# insert an element at index 3
L.insert(3, 50)

# sort the list
L.sort()

# delete a slice (section) of list
del L[4:5]

# reverse the list
L.reverse()
```


chris@simplistix.co.uk
http://www.simplistix.co.uk

Data Types – Strings

- Strings are sequences!
- Handy string methods
  - Capitalise, center, count, startswith, find, isalpha, *etc*
- “string”.join(sequence)
  - The necessary idiosyncracy
- Regular expressions

```
>>> astring = "the quick brown fox"
>>> import re
>>> r = re.compile("((quick|brown)\s)+")
>>> r.findall(astring)
[('brown ', 'brown')]
```


chris@simplistix.co.uk
http://www.simplistix.co.uk

Data Types – Dictionaries

- Associative arrays
- Used for name spaces
- Keys can be anything immutable
- Values can be anything

```
# create a dict
>>> mydict = {'one':1}

# check if it has a key
>>> mydict.has_key('one')
True

# access an item
>>> mydict['one']
1

# set an item
>>> mydict[2]='two'
```


chris@simplistix.co.uk
http://www.simplistix.co.uk

Data Types - Others

- None
  - Indicates nothing
  - Use identity comparison
- Booleans
  - New in Python 2.2
- Objects
  - Can implement special methods to appear like primitive data types
- Complex data structures can be built
  - Just from the primitives!

chris@simplistix.co.uk
http://www.simplistix.co.uk

Conditionals

- Statement structure

```
if expression:
    statement block
elif expression:
    statement block
else:
    statement block
```

- Boolean logic
  - shortcut evaluation
  - returns the result of the last evaluation

```
>>> 0 or 1
1
>>> True and "one thing" or "another"
'one thing'
>>> False and "one thing" or "another"
'another'
```


chris@simplistix.co.uk
http://www.simplistix.co.uk

Conditionals

- What is true?
  - Anything that isn't false:
    - False (d'uh!)
    - None
    - 0
    - [] - empty lists
    - {} - empty dictionaries
    - “” - empty strings
  - Beware the gotchas!

chris@simplistix.co.uk
http://www.simplistix.co.uk

Iterations

- For

```
for item1,item2,itemn in sequence:
    statement block
    if something():
        continue
```

- While

```
while expression:
    statement block

# often see
while 1:
    do_something()
    if something_else():
        break
```

- ranges
  - lets have a play...

chris@simplistix.co.uk
http://www.simplistix.co.uk

Output

- print statement
  - goes to standard output
  - string interpolation

```
>>> x = 1
>>> y = 2
>>> print "x:%.2i" % x
x:01
>>> print "x:%.2i y%2.i" % (x,y)
x:01 y 2
>>> print "x:%(x)i" % {'x':y}
x:2
```

  - See docs for full options...
- pprint

chris@simplistix.co.uk
http://www.simplistix.co.uk

Input

- input from console

```
>>> s = raw_input('--> ')
--> Monty Python's Flying Circus
>>> s
"Monty Python's Flying Circus"
```

- input from command line

```
C:\>python -ic pass one two three
>>> import sys
>>> sys.argv
['-c', 'one', 'two', 'three']
>>>
```


chris@simplistix.co.uk
http://www.simplistix.co.uk

Input and Output

- "stream" like things
  - files
  - sockets
  - StringIO
- Have the following methods
  - open
  - close
  - write
  - seek

```
# read
>>> f = open('numbers.txt')
>>> f.read()
'chris 1234\ndave  5678\njohn
9101\n'
>>> f.close()

# write
>>> f = open('numbers.txt','a')
>>> f.write('bob 2020\n')
>>> f.close()
```


chris@simplistix.co.uk
http://www.simplistix.co.uk

Functions

- Defining functions

```
def donothing():
    pass
a = 1
def adder(b):
    return a + b
```

- What do these do?

```
>>> donothing()
>>> adder(10)
11
>>> a = 12
>>> adder(13)
25
```

- Scoping

chris@simplistix.co.uk
http://www.simplistix.co.uk

Functional Programming

- Lambda
  - anonymous functions
- Map
  - apply a function to a sequence
- Zip
  - combine two sequences

```
>>> x = lambda z: z+1
>>> x(2)
3
>>> y = range(0,5)
>>> map(x,y)
[1, 2, 3, 4, 5]
>>> z = range(6,11)
>>> zip(y,z)
[(0, 6), (1, 7), (2, 8), (3, 9), (4, 10)]
```


chris@simplistix.co.uk
http://www.simplistix.co.uk

Classes

- Every object has
  - data (attributes)
  - a recipe for how that data may be used
- The class is the recipe
  - class attributes
  - methods

```
class sample:

    x = 1

    def do(self,y):
        self.x += y
```

```
>>> o = sample()
>>> o.x
1
>>> o.do(3)
>>> o.x
4
```


chris@simplistix.co.uk
http://www.simplistix.co.uk

Classes

- Inheritence
  - sharing code, minimising duplication

```
class angry:

    def growl(self):
        print "grrrr"

class fuzzy:

    def stroke(self):
        print "prrr"

class bear(angry,fuzzy):

    pass
```

```
>>> ben = bear()
>>> ben.growl()
grrrr
>>> ben.stroke()
prrr
```

  - What happens if two classes define the same method?

chris@simplistix.co.uk
http://www.simplistix.co.uk

Classes

- Class can define “special” methods
  - __init__
    - called to construct the object
  - __call__
    - allows the object to be called

```
class sample:

    def __init__(self,x):
        self.x = x

    def __call__(self,y):
        return self.x * y
```

```
>>> x = sample(2)
>>> x(3)
6
```

- See documentation for full list…

chris@simplistix.co.uk
http://www.simplistix.co.uk

Scripts, Modules & Packages

- Python executes a file from top to bottom
  - a “script” can be run by feeding it to python

- A script can check if it was run from the command line

test.py

```
if __name__=”__main__”:
    print “we have been run as a script”
```

```
C:\>python test.py
we have been run as a script
```


chris@simplistix.co.uk
http://www.simplistix.co.uk

Modules

- A module collects code into a file
- The import statement
  - executes the file
  - makes names available

test.py

```
def my_func():
    print "test"
```

```
>>> import test
>>> test.my_func()
test

>>> from test import my_func
>>> my_func()
test
```


chris@simplistix.co.uk
http://www.simplistix.co.uk

Packages

- A Package is a folder containing
  - a file called __init__.py
  - potentially, more python modules
- Packages can be nested
- Must be imported before being used

testpackage/__init__.py

```
print "importing"
```

testpackage/test.py

```
def my_func():
    print "test"
```

```
>>> import testpackage.test
importing
>>> testpackage.test.my_func()
test
```


chris@simplistix.co.uk
http://www.simplistix.co.uk

Handling Errors - Exceptions

- If something goes wrong, an exception is raised
- Your own exceptions can be defined
- You can raise exceptions when required

```
class MyError(Exception): pass

try:
    x = make_something()
     if not x:
        raise MyError, "no X!"
except MyError,e:
    print e
except:
    print "Unexpected Error"
else:
    print "No error"
```


chris@simplistix.co.uk
http://www.simplistix.co.uk

Debugging

- print
- raise
- tracebacks
- Introspection
  - Everything is 1st class
  - the dir function
- Debugger
  - lets see an example…

chris@simplistix.co.uk
http://www.simplistix.co.uk

The Example

- Now lets review the example…

chris@simplistix.co.uk
http://www.simplistix.co.uk

Resources

- http://www.python.org
- irc.freenode.net #python
- documentation is distribution
- help function

chris@simplistix.co.uk
http://www.simplistix.co.uk

Epilogue

- Jython
- C/C++ extensions
- Platform specific libraries
- Embeddable – games!

chris@simplistix.co.uk
http://www.simplistix.co.uk

How do I do that?

- Fire away…

chris@simplistix.co.uk
http://www.simplistix.co.uk

Thankyou!

- Chris Withers
- chris@simplistix.co.uk
- http://www.simplistix.co.uk
- Do people want these slides to be available?