Introduction to Python

Virtual Environments
Spring 2008
Reykjavík University

Adapted with gratitute from Brad Dayley's "Python Phrasebook" and CSE 391 slides at UPenn

What is Python

- Powerful open source scripting language.
- Developed by Guido van Rossum in early 90s.
- Named after Monty Python.
- Maintained at:

  www.python.org

Why Python?

- Portability
  - Interpreters available on almost any platform.
- Integration
  - Can contain C/C++ code. Can run on Java.
- Ease of use
  - Clear and readable syntax. Intuitive data types.
- Power
  - Powerful extensions added all the time.
- Dynamic
  - Flexible language that supports creative solutions.
- Open Source
  - Free to use and distribute.

Who Uses Python?

- Examples:
  - Google
  - Industrial Light and Magic
  - United Space Alliance
  - Disney (Panda 3D)
  - CCP Games (EVE Online)
  - Sid Meier (Civilization IV)
  - etc.

Invoking The Interpreter

- If the python executable (python.exe on PCs) is in your execution path, just type:
  python
  in any shell to invoke the interpreter in interactive mode.
- The command:
  >>> execfile("scriptfile.py")
  will interpret the contents of this script.
- Quicker to invoke the interpreter with a script parameter:
  python scriptfile.py

Types

- Built-in object types. Type guessed at assignment time. Can determine later whether object is of a specific type:

```
>>> s = "A Simple String"
>>> print isinstance(s, basestring)
True
>>> print isinstance(s, dict)
False
>>>
```

Types

- Built-in object types:
  - General object, type
  - Null Types.NoneType
  - Numbers bool, int, long, complex
  - Sets set, frozenset
  - Sequences str, list, tuple, xrange
  - Maps dict
  - Files file
  - Callable types.FunctionType, types.MethodType

Types

- Numbers
  - **bool** is either True or False
    ```
    >>> x = True
    ```
  - **int** is 32 bit whole numbers while long is only limited by machine memory.
    ```
    >>> x = 4
    ```
  - **float** is 64 bit floating-point numbers.
    ```
    >>> x = 4.3
    ```
  - **complex** is a pair of floats.
    ```
    >>> x = 1.5+0.5j
    >>> print x.real, x.imag
    1.5  0.5
    ```

Types

- Sets
  - An unordered collection of unique items.
  - Mutable sets can be modified.
  - Immutable sets cannot be changed after creation!

Types

- Sequences
  - Ordered items, indexed by integers.
  - Can be made up of almost any Python object.
  - **str**ings and **tuple**s are immutable.
    >>> mystring = "hello"
    >>> mytuple= (1, mystring, 3.5)
  - **list**s are mutable.
    >>> mylist = [1, "hello", 3.5]
    >>> mylist[1] = "bye"

Types

- Sequences: Indexing
  - Typical array notation starting with 0, also negative indexing from right starting with -1.

    >>> mystring[4]
    'o'
    >>> mystring[-1]
    'o'

Types

- Sequences: Slicing
  - Returning a copy with a subset of original sequence. Start copying at first index and stop copying before second index.

    >>> mystring[2:4]
    'll'
    >>> mytuple[0:-1]
    (1, 'hello')

Types

- Maps (i.e. Dictionary, Hash tables, Associative Arrays)
  - A collection of key objects that index the second collection of value objects.
  - The key object must be of an immutable type.
  - The value object can be almost any Python object.

```
>>> trans = {'epli':'apple' , 'appelsína':'orange'}
>>> trans['epli']
'apple'
```

Types

- Files
  - Object representing an open file.
  - Used to read and write filesystem data.
- Callables
  - Objects of this type can be called as a function.
  - For example built-in functions, user-defined functions and method instances.

Types

- Modules
  - Modules of code loaded with the import statement.
  - All objects within a module can be accessed using the dot syntax.

```
>>> import math
>>> print math.pi
3.14159265359
```

Syntax

- Code indentation
  - There are no { } or begin/end markers for code blocks.
  - Blocks of code are denoted by line indentation.
  - Number of spaces may vary across blocks, but never within a block!

```
if True:
   print "Good answer:"
   print "True"
else:
    print "Bad answer:"
   print "Fale"
```

Syntax

- Multiline Statements
  - Statements end with a new line.
  - Can use \ to denote the line continues.

```
Sum = x + 4 \
        5.6 + y
```

  - Statements within [], {} or () don't need this.

```
List =  ['apple', 'orange',
         'lemon', 'pear']
```

Syntax

- Quotation
  - Single ('), double ("), triple ('' or """).
  - Have to match at each end.
  - Triple quotes can span multiple lines.

```
s = 'hello'
s = "hello again 'sam'! "
s = """hello! What I meant to say was
                how are you doing?"""
```

Syntax

- Comments
  - **#** starts a comment to the end of the line
  - "Documentation strings" can be included as the first line of any new class or function definition

```
def foo(x, y):
    """Does foo to both x and y
        blah blah blah """
    # Now the code starts
    print x, y
```

Syntax

- Formatting strings
  - Match a list of objects to predefined format symbols within a string.

```
>>> X = ["Sam", 1]
>>> print "%s is number %03d%s" % (x[0], x[1], "!")
Sam is number 001!
```

Syntax

- Flow Control
  - **if** expression**:** block
  - **while** expression**:** block
  - **for** item **in** sequence**:** block
  - **else** and **elif** added to any of these.
  - **break** exits a loop (skips an else), **continue** jumps to next iteration.

Objects, Classes and Functions

- Objects
  - Every piece of data stored and used in the Python Language is an **object**.
  - Every object has
    - **Identity**: points to memory location
    - **Type**: describes object representation / interpretation
    - **Value**: the data

```
>>> x = 3
>>> print id(x), type(x), x
10115944 <type 'int'> 3
```

Objects, Classes and Functions

- Objects cont.
  - Can also have
    - **Attributes**: Other values associated with the object.
    - **Methods**: Callable functions associated with the object.
    - Those are accessed with the dot-notation.

```
>>> class foo(object):
        def p(self):
            print self.num
>>> f = foo()
>>> f.num = 3
>>> f.p()
3
```

Objects, Classes and Functions

- Classes
  - Basically a collection of attributes and methods.
  - **"class** name**(***object***):** block" defines a new class that derives from *object*.
  - All code contained in the block will be executed when the class is instantiated.
  - The **"__init__()**" function (method) will also be executed if defined inside the block (constructor).

Objects, Classes and Functions

- Classes cont.

```
class testClass(object):
        print "Creating a testClass instance"
        number = 5
        def __init__(self, string):
                self.string = string
        def print(self):
                print "Number=%d" % self.number
                print "String=%s" % self.string

tc = testClass("Five")
tc.print()
tc.number = 10
tc.string = "Ten"
tc.print()
```

OUTPUT:
Creating a testClass instance
Number = 5
String = Five
Number = 10
String = Ten

Objects, Classes and Functions

- Functions
  - Functions are objects in Python.
  - "**def** functionname**(**parameters**):** block" defines a new function.
  - Parameters are not type checked!
  - Parameters can be passed in a number of ways.

Objects, Classes and Functions

- Functions Cont.

```
def fun(name, location, year=2006):
  print "%s/%s/%d"%(name,location,year)
```

```
>>> fun("Teag", "San Diego")
Teag/San Diego/2006
```

Objects, Classes and Functions

- Functions Cont.

```
def fun(name, location, year=2006):
  print "%s/%s/%d"%(name,location,year)

>>> fun(location="San Diego", name="Teag",
        year=2004)
    Teag/San Diego/2004
```

Objects, Classes and Functions

- Functions Cont.

```
def fun(name, location, year=2006):
  print "%s/%s/%d"%(name,location,year)

>>> fun("Teag", year=2004,
        location="San Diego")
    Teag/San Diego/2004
```

Objects, Classes and Functions

- Functions Cont.

```
def fun(name, location, year=2006):
  print "%s/%s/%d"%(name,location,year)

>>> tuple = ("Teag","San Diego",2004)
>>> fun(*tuple)
    Teag/San Diego/2004
```

Objects, Classes and Functions

- Functions Cont.

```
def fun(name, location, year=2006):
  print "%s/%s/%d"%(name,location,year)


>>> dictionary = {name="Teag",
          location="San Diego", year=2004}
>>> fun(**dictionary)
    Teag/San Diego/2004
```

Objects, Classes and Functions

- Functions Cont.
  - Values can be returned from functions using the **return** statement.
  - If a function has no **return** statement, a **None** object is returned.

```
>>> def square(x):
        return x*x
>>> print square(3)
9
```