Introduction to Python Programming

CS 360 Internet Programming

Daniel Zappala

Brigham Young University
Computer Science Department

Hello World

```
1 print "Hello World!"
```

Hello World Function

```
1  def hello_world():
2      print "Hello World!"
3
4  hello_world()
```

Hello World Class

```
1  class HelloWorld:
2      def greet(self):
3          print "Hello World!"
4
5  h = HelloWorld()
6  h.greet()
```

Language Features

- interpreted
- interactive
- object-oriented
- simple, human-readable syntax
- easy to integrate with C, C++
- modules, classes, exceptions, high level data types
- dynamically typed - types are discovered at runtime
- strongly typed - types are always enforced, you must explicitly convert types

Code Indentation

- code blocks are determined by indentation
- the only delimiter is ":"

```
1
2 def greet():
3     print "Hello"
4 print "  World!"
5
6 greet()
```

Variables and Types

- variable names
  - must start with a letter
  - may contain numbers, underscore
  - are case-sensitive
  - may not be a keyword
- automatically created when assigned, destroyed when out of scope

```
1  >>> message = "How are you?"
2  >>> n = 17
3  >>> pi = 3.14159
4  >>> a = False
5  >>> type(message)
6  <type 'str'>
7  >>> type(n)
8  <type 'int'>
```

Expressions

- combine values, variables, and operators
- Python includes common math operators, functions

```
1  20+32
2  hour-1
3  hour*60+minute
4  minute/60
5  minute/60.0
6  5**2
7  (5+9)*(15-7)
```

Statements and Comments

- statements
  - print, assignment, etc
  - ended by a newline
  - continued by " /"
- comments
  - started by "#"
  - can be at the end of a line

```
1  # compute the percentage of the hour that has elapsed
2  percentage = (minute*100)/60     # integer division
```

Lists

- can contain arbitrary objects
- can expand dynamically as objects are added
- *many* convenient list operations

```
1  >>> l = ['a','b','gorilla','z',1]
2  >>> l[0]
3   'a'
4  >>> l[2]
5  "gorilla"
6  >>> l[-2]
7   'z'
```

List Operations

```
>>> l = ['a','b','gorilla','z',1]
>>> l[0:3]                                  # slicing
['a','b','gorilla']
>>> l[3:]
['z',1]
>>> l.append('new')                         # append
>>> l
['a','b','gorilla','z',1,'new']
>>> l.insert(2,'again')                     # insert
>>> l
['a','b','again','gorilla','z',1,'new']
>>> l.extend(['second','third'])            # extend
>>> l
['a','b','again','gorilla','z',1,'new','second','third']
>>> len(l)                                  # length
9
```

More List Operations

```
>>> l = ['a','b','gorilla','z',1]
>>> l.index('gorilla')                    # searching
2
>>> 'church' in l                         # membership
False
>>> l.remove('z')                         # remove
>>> l
['a','b','gorilla',1]
>>> l.pop()                               # pop
1
>>> l
['a','b','gorilla']
>>> l += ['more','items']                 # add
>>> l
['a','b','gorilla','more','items']
>>> l = ['a','b']
>>> l*2                                   # multiply
['a','b','a','b']
```

Dictionaries

- mapping of keys to values
- keys must be unique
- assigning a new value to a key erases the old value
- keys can be strings or integers
- values can be any type or data structure

```
1  >>> d = {'Smith':'A','Li':'B+','Students':2}
2  >>> d['Smith']
3   'A'
4  >>> d['Anderson'] = 'C'                    # add key
5  >>> d['Students'] = 3                      # change value
6  >>> d
7  {'Students':3,'Anderson':'C','Smith':'A','Li':'B+'}
```

More on Dictionaries

```
>>> del d['Smith']                    # delete
>>> d
{'Students': 3, 'Anderson': 'C', 'Li': 'B+'}
>>> d.keys()                          # list keys
['Students', 'Anderson', 'Li']
>>> d.values()                        # list values
[3, 'C', 'B+']
>>> d.items()                         # list item tuples
[('Students', 3), ('Anderson', 'C'), ('Li', 'B+')]
>>> d.has_key('Jones')                # key existence
False
>>> d.clear()                         # clear dictionary
>>> d
{}
```

Tuples

- an immutable list

```
>>> t = ('a','b','gorilla','z',1)
>>> t[0]
 'a'
>>> t[-1]
 1
>>> t[-3:]
 ('gorilla', 'z', 1)
>>> 'z' in t
 True
```

Strings

- immutable sequence of characters
- special characters: $\backslash n$ $\backslash t$
- surround with matching double or single quotes
- formatting like sprintf in C

```
1  >>> s = "hello"
2  >>> s[1]
3   'e'
4  >>> s + " world"                    # concatenation
5   'hello world'
6  >>> len(s)                          # length
7   5
8  >>> n = 1
9  >>> print "%d. %s " % (n,s)         # formatting
10  1. hello
```

String Methods

```
>>>
>>> s.find('o')                                    # search
4
>>> s.upper()                                      # uppercase
'HELLO'
>>> s.replace('e','i').replace('l','p')
'hippo'                                            # replace
>>> s = "The quick brown fox"
>>> s.split()                                      # split
['The', 'quick', 'brown', 'fox']
>>> l = ['jumped','over','the','lazy','dog']
>>> " ".join(l)                                    # join
'jumped over the lazy dog'
>>> s + " " + " ".join(l)
'The quick brown fox jumped over the lazy dog'
```

Defining a Method

- declare arguments, some of which can be optional
- you may return any value or object
- default return value is NULL

```
1  >>> def increment(value,step=1):
2  ...         value += step
3  ...         return value
4  ...
5  >>> increment(1)
6  2
7  >>> increment(5,2)
8  7
```

Defining a Class

- may initialize the class in __init__ method
- all methods must have "self" as the first argument

```
>>> class Number:
...     def __init__(self,value=0):
...         self.value = value
...     def increment(self,step=1):
...         self.value += step
...     def value(self):
...         return self.value
...
>>> n = Number()
>>> n.value()
0
>>> n.increment(2)
>>> n.value()
2
```

Importing Modules

- import the module to call its functions within its namespace
- import individual methods from the module (can use wildcard)
- import search path is given by sys.path (just a list of directories)

```
1  >>> import math
2  >>> math.sqrt(9)
3   3.0
4  >>> from math import sqrt
5  >>> sqrt(9)
6   3.0
7  >>> import sys
8  >>> sys.path
9   ['/usr/lib/python2.4/site-packages','/usr/lib/python2.4',...]
```

Exporting all Methods in a Module

```
__all__ = ["Tree","Node","Hash"]

class Tree:
    ...

class Node:
    ...

class Hash:
        ...
```

```
from datastructs import *

t = Tree()
h = Hash()
```

Exceptions

- familiar try/except syntax
- if exception is caught, handle it
- execution continues after the except block

```
1  try:
2      fsock = open("/notthere")
3  except IOError:
4      print "The file does not exist, exiting gracefully"
5  print "This line will always print"
```

Raising Exceptions

```
>>> class MyError(Exception):
...     def __init__(self, value):
...         self.value = value
...     def __str__(self):
...         return repr(self.value)

>>> try:
...     raise MyError(2*2)
... except MyError as e:
...     print 'My exception occurred, value:', e.value

My exception occurred, value: 4
```