Introduction to Python Programming

Daniel Zappala

CS 360 Internet Programming
Brigham Young University
Hello World

```python
1 print "Hello World!"
```
Hello World Function

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

```python
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 "::

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

```python
>>> message = "How are you?"
>>> n = 17
>>> pi = 3.14159
>>> a = False
>>> type(message)
<type 'str'>
>>> type(n)
<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

```python
>>> l = ['a', 'b', 'gorilla', 'z', 1]
>>> l[0]
'a'
>>> l[2]
'gorilla'
>>> l[-2]
'z'
```
List Operations

```python
1 >>> l = ['a', 'b', 'gorilla', 'z', 1]
2 >>> l[0:3] # slicing
3 ['a', 'b', 'gorilla']
4 >>> l[3:]
5 ['z', 1]
6 >>> l.append('new') # append
7 >>> l
8 ['a', 'b', 'gorilla', 'z', 1, 'new']
9 >>> l.insert(2, 'again') # insert
10 >>> l
11 ['a', 'b', 'again', 'gorilla', 'z', 1, 'new']
12 >>> l.extend(['second', 'third']) # extend
13 >>> l
14 ['a', 'b', 'again', 'gorilla', 'z', 1, 'new', 'second', 'third']
15 >>> len(l) # length
16 9
```
More List Operations

1 >>> l = ['a','b','gorilla','z',1]  # searching
2 >>> l.index('gorilla')  
3 2
4 >>> 'church' in l  # membership
5 False
6 >>> l.remove('z')  # remove
7 >>> l  
8 ['a','b','gorilla',1]
9 >>> l.pop()  # pop
10 1
11 >>> l  
12 ['a','b','gorilla']
13 >>> l += ['more','items']  # add
14 >>> l  
15 ['a','b','gorilla','more','items']
16 >>> l = ['a','b']
17 >>> l*2  # multiply
18 ['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

```python
>>> d = {'Smith': 'A', 'Li': 'B+', 'Students': 2}
>>> d['Smith']
'A'
>>> d['Anderson'] = 'C'  # add key
>>> d['Students'] = 3    # change value
>>> d
{'Students': 3, 'Anderson': 'C', 'Smith': 'A', 'Li': 'B+'}
```
More on Dictionaries

```python
>>> del d['Smith']  # delete
>>> d
{}
```

```python
>>> d = {'Students': 3, 'Anderson': 'C', 'Li': 'B+'}
>>> d.keys()  # list keys
['Students', 'Anderson', 'Li']

```python
```python
>>> d.values()  # list values
[3, 'C', 'B+']

```python
```python
>>> d.items()  # list item tuples
[('Students', 3), ('Anderson', 'C'), ('Li', 'B+')]

```python
```python
>>> d.has_key('Jones')  # key existence
False

```python
```python
>>> d.clear()  # clear dictionary
>>> d
{}
```
Tuples

- an immutable list

```python
>>> 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: `\n \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

```python
>>> s.find('o')  # search
4

>>> s.upper()  # uppercase
'HELLO'

>>> s.replace('e', 'i').replace('l', 'p')  # replace
'hippo'

'Hello'

>>> 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'
'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

```python
>>> def increment(value, step=1):
    value += step
    return value

>>> increment(1)
2
>>> increment(5, 2)
7
```
**Defining a Class**

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

```python
>>> 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)

```
>>> import math
>>> math.sqrt(9)
3.0
>>> from math import sqrt
>>> sqrt(9)
3.0
>>> import sys
>>> sys.path
['/usr/lib/python2.4/site-packages', '/usr/lib/python2.4', ...]
```
Exporting all Methods in a Module

```python
    __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

```python
try:
    fsock = open("/notthere")
except IOError:
    print "The file does not exist, exiting gracefully"
print "This line will always print"
```
Raising Exceptions

```python
>>> 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
```