Python Tutorial

CSE 3461: Computer Networking

Outline

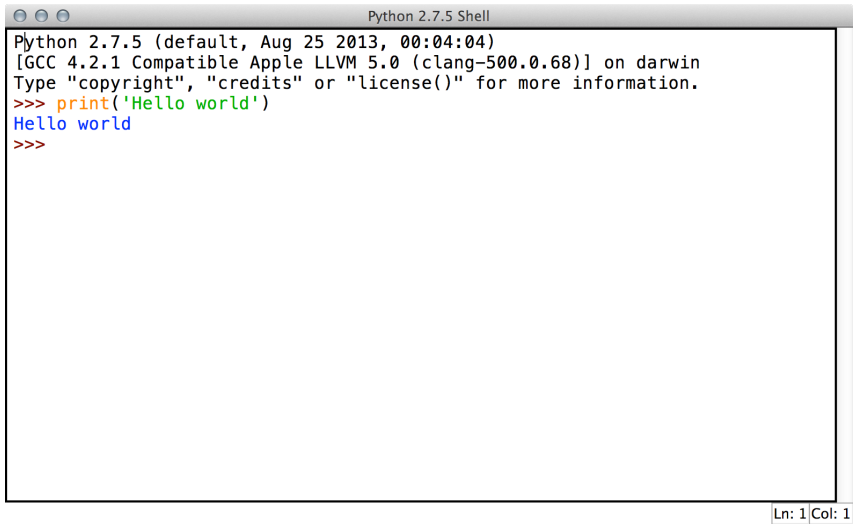
- **Introduction to Python**
- CSE Environment
- Tips for Python Primitive Types
- Tips for Encoding/Decoding an IP Address

Intro to Python

- Dynamically typed, object-oriented, interpreted scripting language
  - Not statically typed like Java
  - Objects and exceptions similar to Java
  - Concise style (in contrast to C!)
  - Interpreted via interpreter vs. compilation, linking, and execution
- Python 3.x breaks backward compatibility with 2.x
  - Not all libraries with 2.x work with 3.x (e.g., twister)
  - But 3.x offers features not found in 2.x…
- Many Python references online, including:
  - Python tutorial: http://docs.python.org/3/tutorial/index.html
  - N.R. Ceder, *The Quick Python Book*, 2nd ed., Manning, 2010, http://proquest.safaribooksonline.com/book/programming/python/9781935182207

Running Python Programs

- Interpreters:
  - Interactive mode (type 'python' at command line)
  - IDLE CSE Environment (type 'idle' at command line)
- Scripts
  - Create a file beginning with: `#!/usr/bin/env python`
  - Then add your code



helloworld.py

```
#!/usr/bin/env python
print 'Hello world'
```

```
$ python helloworld.py
Hello world
```

Running 'Hello World' in IDLE (above) and as a script (below).

Basic Data Types (1)

- Numbers:
  - Integers (`-3`, `0`, `5`)
  - Floats (`3.0`, `-6.0`, `2.5e12`, `-2.5e-12`)
  - Complex numbers (`3+2j`, `-3-2j`)
  - Booleans (`True`, `False`)
- Strings: *immutable* sequences of characters, indices start at 0
  - If `a = 'Python'`, then `a[0]` returns `'P'`, `a[5]` returns `'n'`
  - Positive and negative indices:

```
 +---+---+---+---+---+---+
 | P | y | t | h | o | n |
 +---+---+---+---+---+---+
 0   1   2   3   4   5   6
-6  -5  -4  -3  -2  -1
```

  - Slicing: `a[i:j]` returns substring of a containing characters `i`, `i+1`, … `j-1` (e.g., `a[2:4]` returns "`th`")
  - `str()` converts an "object" to a string, e.g., `str(5)` returns "`5`"

Basic Data Types (2)

- Lists: *mutable* sequences of "objects"
  - Example: `[1,2,3]` and `[1,'two',3]`
  - List elements can be changed:
    if `a = [1,5,9]`, `a[1] = 4` yields `a = [1,4,9]`
  - Operators: `len()`, `max()`, `min()`, `append()`, `count()`, `extend()`, `index()`, `insert()`, `pop()`, `remove()`, `reverse()`, `sort()`, `in`, `+`, `*`
  - `len()` also returns length of string
  - `list()` constructs a list from its input
- Tuples: *immutable* "vectors" of objects (keys in dictionaries)
  - Example: `(1,)`, `(1,2)`, and `(1, 'two', 3)`
  - Operators `in`, `+`, `*`, `len()`, `max()`, `min()` apply
  - `tuple()` and `list()` convert lists to tuples and vice versa

Basic Data Types (3)

- Dictionaries: maps between *immutable* keys and *mutable* values
  - Ex: `x={“one”:1,“two”:2}` and `[“three”]=3` yield `x={“one”:1,“two”:2,“three”:3}`
  - Operators: `len()`, `del()`, `clear()`, `copy()`, `get()`, `has_key()`, `items()`, `keys()`, `update()`, `values()`
- File objects: file I/O is very simple
  - Ex:
```
f = open(“file.txt”, “r”)
line = f.readline()
print(line)
```

Control Structures (1)

- Boolean connectives are mostly the same as other languages (`>`, `≥`, `<`, `≤`, `==`, `!=`)
  - Specialties: `and`, `or`, `not`, `is`, `is not`, `in`, `not in`
- If-then-else: the "else if" is `elif`:
  - ```
    x = 5
    if x < 5:
        x = x + 1
    elif x > 5:
        x = x - 1
    else:
        print(x)
    ```
  - What happens?
  - Notice *indentation* determines control structure "level"; no braces! Usually four spaces (no tabs)
    ```

Control Structures (2)

- while loop: executes as long as stmt. is true
  - Example:

```
x = 5
while (x > 0):
    x = x - 1
print(x)
```

- for loop: iterates over "iterable" objects…
  - Example:

```
alist = list([1, 2, 3, 4])
for item in alist:
    print(item)
```

Function Definition

- Python lets us define our own functions
  - Ex:

```
def find_mean(iterable):
    the_sum = 0
    for x in iterable:
        the_sum = the_sum + x
    the_sum = float(the_sum / len(iterable))
    return the_sum
a = list([1,2,3,4])
mu_a = find_mean(a)
print(mu_a)
```

Class Definition

- Python enables object-oriented programming:
- Ex. (Listing 3.3 in *The Quick Python Book*):

```
"""sh module. Contains classes Shape, Square and Circle"""
class Shape:                                                    (1)
    """Shape class: has method move"""
    def __init__(self, x, y):                                   (2)
        self.x = x                                              (3)
        self.y = y
    def move(self, deltaX, deltaY):                                 (4)
        self.x = self.x + deltaX
        self.y = self.y + deltaY
class Square(Shape):
    """Square Class:inherits from Shape"""
    def __init__(self, side=1, x=0, y=0):
        Shape.__init__(self, x, y)
        self.side = side                                        (5)
class Circle(Shape):
    """Circle Class: inherits from Shape and has method area"""
    pi = 3.14159                                                (6)
    def __init__(self, r=1, x=0, y=0):
        Shape.__init__(self, x, y)                                  (7)
        self.radius = r
    def area(self):
        """Circle area method: returns the area of the circle."""
        return self.radius * self.radius * self.pi
    def __str__(self):                                              (8)
        return "Circle of radius %s at coordinates (%d, %d)"\
               % (self.radius, self.x, self.y)
```

Outline

- Introduction to Python
- **CSE Environment**
- Tips for Python Primitive Types
- Tips for Encoding/Decoding an IP Address

CSE Environment (1)

- Both Python 2.x or 3.x are available on `stdlinux`
- The default version of Python on `stdlinux` is 2.6.6
- To use Python 3.2.3, please use **`subscribe`** command
  - On `stdlinux`, type **`subscribe`** and select ***`PYTHON-3`***
  - Then, log out from and log in again to `stdlinux`
  - Please make sure that `python3.x` is installed with `python3 -v`
- The execution commands are
  - **`python`** for 2.6.6
  - **`python3`** for 3.2.3

CSE Environment (2)

- How to find the IP address that you are logging in
  - `/sbin/ifconfig`
- Submission command
  - `submit c3461a`*`x`*` lab1 [code1] [code2] …`
  - Where *x* could be `a`, `b`, and so on
  - Note that the last submission overwrites the previous submission

Tips for Python Primitive Types

- Introduction to Python
- CSE Environment
- **Tips for Python Primitive Types**
- Tips for Encoding/Decoding an IP Address

Encoding Integer to Bytes (Python 2.x)

- For version 2.x, you can use "`struct`" class
- Note that "`<`" means little endian, and "`I`" means unsigned integer
- An integer variable is always 4 bytes, and thus the `pack()` function returns a byte object with size 4

```
>> import struct
>> var = 1000          # an integer variable
>> byte_int = struct.pack('<I', var) # a byte object
```

Encoding Integer to Bytes (Python 3.x)

- For version 3.x, there's a simple way.
- Note that `(int).to_bytes(int, byteorder)` was introduced in 3.1; not available for 2.6.6.
- The first argument is the number of bytes, and the second argument is '`little`', '`big`', etc.

```
# var is encoded into a byte object, byte_int, with size 4
# with little endian.
>> var = 1000          # an integer variable
>> byte_int = var.to_bytes(4, byteorder='little')
```

Decoding Byte to Integer (Python 2.x)

- Use "`struct.unpack`"
- `struct.unpack('<I', bytes_var)` returns a tuple; first element is an integer value
- "`<`" indicates the little endian, and "`I`" indicates unsigned integer type

```
# byte_int is a byte object with size 4.
>> import struct
>> var = struct.unpack('<I', byte_int)[0]
```

Decoding Byte to Integer (Python 3.x)

- Use "`int.from_bytes(bytes_var, byteorder)`"
- This is a static function, thus in form of `int.function(...)`

```
 # byte_int is a byte object
>> var = int.from_bytes(byte_int, byteorder='little')
```

Encoding String to Bytes

- String type has `encode()` and `decode()` functions
- ASCII is default encoding scheme

```
>> str_var = "Hello.”
>> byte_var = str_var.encode()  # encoding with ASCII
>> str_var2 = byte_var.decode() # decoding with ASCII
```

Formatting String (Constant Length)

- Use `(str_var).rjust(int)`, where the argument is the length after formatting
- Spaces are added in keeping with the original string at the right (the end of the string)
- Similar functions are available, such as "`ljust(int)`", and so on

```
# e.g., "Hello" (5-byte) is formatted to "     Hello" (10-byte)
>> str_var = "Hello”
>> formatted_str_var = str_var.rjust(10)
```

Removing Spaces from String

- Use "`(str_var).lstrip()`" that removes spaces from the left (the beginning of the string), where `str_var` is a string object

```
# e.g., The spaces in "     Hello" are removed from the left
# (the beginning of the string),
# and lstrip() reutrns "Hello".
>> formatted_str_var = "     Hello”
>> str_var = formatted_str_var.lstrip()
```

Outline

- Introduction to Python
- CSE Environment
- Tips for Python Primitive Types
- **Tips for Encoding/Decoding an IP Address**

Encoding Integer to Byte (Python 2.x)

- Note that Python 3.x is recommended…
- For Python 2.x, you can encode an IP address as follows: for each integer value, `[127, 0, 0, 1]`, pack integer as a binary with the `'B'` option. "Reverse" operation for decoding.

```
# Note that "var" must be between 0 - 255.
>> import struct
>> var = 3
>> byte_var = struct.pack('B', var)
>> \x03
```

```
# unpacking, where byte_var is \x03
>> unpacked_var = struct.unpack('B', byte_var)
>> 3
```

Encoding Integer to Byte (Python 3.x)

- Encode an IP address into bytes with size 4 as follows:
  - Split the string object "`127.0.0.1`" to a list with four elements, i.e., [`"127”,"0”,"0”,"1"`], where "`.`" is token
  - Cast every element to an integer value, i.e., [`127,0,0,1`]
  - Use "`to_bytes`" function for each integer value
  - Concatenate bytes with “+”.

```
# byte_var is a 1-byte byte object.
# The first argument of "to_byte" is the size of a byte object.
>> var = 10
>> byte_var = var.to_bytes(1, byteorder='little')
```

```
# byte_var3 is "\x9b\x00"
>> byte_var1 = \x9b
>> byte_var2 = \x00
>> byte_var3 = byte_var1 + byte_var2
```

Acknowledgment

- This material is partially based on

  http://www.seas.upenn.edu/~cis521/Lectures/python-tutorial.pdf

  N.R. Ceder, *The Quick Python Book*, 2nd ed., Manning Publications, 2010.

  Python Tutorial, http://docs.python.org/3/tutorial/introduction.html