

Introduction to Python

Mubashir Adnan Qureshi

What is Python?

- ❑ Designed by Guido van Rossum, in 1990s
- ❑ Dynamic, interpreted language
 - ❖ does not declare types of variables or parameters
 - ❖ short and flexible code
 - ❖ no compile-time type checking
- ❑ Good for fast prototyping

Python Interpreter

- ❑ Installed on every cs machine

```
aero$ python
Python 2.7.3 (default, Feb 27 2014, 19:58:35)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more informat
ion.
>>> 
```

```
>>> l = [i*i for i in range(15)]
>>> l
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196]
>>> sum(l)
1015
```

- ❖ ctrl-D to exit or use exit()

Python Scripts

- Python scripts suffix with .py

- In general,

 - ❖ `$python ./helloworld.py`

- Create executable scripts

 - ❖ in the first line of the script, add

 - `#!/usr/bin/python`

 - ❖ make the file executable, type in the terminal

 - `$chmod a+x helloworld.py`

 - ❖ type the name of the script to execute

 - `$/helloworld.py`

Syntax

- ❑ Much of Python syntax is similar to C
- ❑ Missing operators: `++`, `--`
- ❑ Code blocks denoted by line indentation
 - ❖ class and function definitions, control flow
 - ❖ same amount of indentation within the same block

```
if True:  
    print 'true'  
    print 'answer'
```

```
if True:  
    print 'true'  
    print 'answer'
```

```
if True:  
    print 'true'  
Else:  
    print 'false'
```

- ❑ Hash sign (`#`) begins a comment

Built-in Types

□ Numerical types

- ❖ integer, float, complex
- ❖ bitwise operations on integer types are the same as in C

□ Sequence types

- ❖ string, list, tuple, bytearray, xrange, buffer, ...

□ Mapping type – dictionary

- ❖ dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'};
- ❖ dict['Name'] = ? dict['Age'] = ?

String Methods

❑ Concatenation

❖ “hello” + “world” “helloworld”

❑ Repetition

❖ “hello”*3 “hellohellohello”

❑ Length

❖ len(“hello”) 5

❑ Indexing

❖ “hello”[2], “hello”[-1]

❑ Slice

❖ “hello”[1:3], “hello”[-2:]

❖ “hello”[:], “hello”[1:10]

He1lo

0	1	2	3	4
-5	-4	-3	-2	-1

String Methods (Cont.)

`s = "hello+world"`

- ❑ `s.find("world")` 6
- ❑ `s.split("+")` ["hello","world"]
- ❑ `"+" .join(["hello","world"])` "hello+world"
- ❑ `"lo" in s` ("lo" not in s) True
- ❑ `s.upper()` (`s.lower()`) "HELLO+WORLD"
- ❑ `str(3.14)` "3.14"

List Methods

- A compound data type

- ❖ `[1, "hello", True, 3.2]`

- Same operators as for strings

- ❖ `a+b, a*3, a[-1], a[1:], len(a)`

`items = [1, "hello", 9.2, True]`

- Append an element

- ❖ `items.append("world")` `[1, "hello", 9.2, True, "world"]`

- Extend the list

- ❖ `items.extend(["world"])` `[1, "hello", 9.2, True, "world"]`

Lists Methods (Cont.)

`items = [1, "hello", 9.2, True]`

❑ Insert an element

❖ `items.insert(2, "world")` `[1, "hello", "world", 9.2, True]`

❑ Remove an element

❖ `items.remove("hello")` `[1, 9.2, True]`

❖ `items.pop()` `[1, "hello", 9.2]`

❑ Reverse the order of the list

❖ `items.reverse()` `[True, 9.2, "hello", 1]`

❑ Generate a list

❖ `range(5)` `[0, 1, 2, 3, 4]`

❖ `[i*i for i in range(5)]` `[0, 1, 4, 9, 16]`

Control Flow – if/elif/else

```
if a == 0:  
    print "zero!"  
elif a < 0:  
    print "negative!"  
else:  
    print "positive!"
```

- ❑ blocks identified by indentation
- ❑ colon (:) used at end of lines containing control flow keywords

Control Flow – for loop

```
for x in list:  
    do something...
```

Example:

```
a = [3, 1, 4, 1, 5, 9]  
for x in a:  
    print x
```

Functions

❑ Defining functions

- ❖ begins with keyword **def**, function name, and parentheses

```
def inc(x):  
    y = x+1  
    return y
```

- ❖ parameters are placed within parentheses
- ❖ code block starts with colon and indented

❑ Calling functions

```
print inc(3)
```

- ❑ Functions must be defined before they are called

Modules

- files containing Python definitions and statements

 - ❖ code reuse, easier maintenance

- Importing modules

```
from socket import *  
socket(AF_INET, SOCK_STREAM)  
import random  
random.randint(1, 10)
```

Packing Datagrams

□ struct module

❖ `from struct import *`

□ A message has two fields. field 1 has 2 bytes, field 2 has 4 bytes

❖ sender side

```
>>> s = pack('!HL', 12, 100000)
>>> s
'\x00\x0c\x00\x01\x86\xa0'
```

network
order

unsigned
short

unsigned
long

❖ receiver side

```
>>> unpack('!HL', s)
(12, 100000)
```

□ Details in Python documentation

When You Need Help...

- ❑ Google search
 - ❖ 'python list', 'python string uppercase', ...
- ❑ Official Python docs site – docs.python.org
- ❑ Many questions (answers) can be found on StackOverflow
- ❑ Use `help()` inside the Python interpreter
 - ❖ `help(len)`, `help(list)`, ...

Example: TCP client in Python

```
from socket import * ← include Python's socket library
serverName = 'servername'
serverPort = 12000
clientSocket = socket(AF_INET, ← create TCP socket
                        SOCK_STREAM)
clientSocket.connect((serverName,serverPort))
sentence = raw_input('Input lowercase sentence:')
clientSocket.send(sentence.encode()) ← No need to attach server name, port
modifiedSentence = clientSocket.recv(1024)
print ('From Server:', modifiedSentence.decode())
clientSocket.close()
```

Example: TCP server in Python

```
from socket import *
```

```
serverPort = 12000
```

create TCP
welcoming
socket

```
serverSocket = socket(AF_INET,  
                      SOCK_STREAM)
```

```
serverSocket.bind(('',serverPort))
```

server begins
listening for
incoming TCP
requests

```
serverSocket.listen(1)
```

```
print 'The server is ready to receive'
```

```
while 1:
```

server waits on
accept()
for incoming
requests, new
socket created on
return

```
connectionSocket, addr = serverSocket.accept()  
sentence = connectionSocket.recv(1024).decode()
```

read bytes from
connection
socket

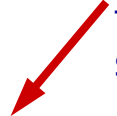
```
capitalizedSentence = sentence.upper()
```

```
connectionSocket.send(capitalizedSentence.encode())
```

```
connectionSocket.close()
```

close connection to this
client (but *not*
welcoming socket)

Example: UDP client in Python

```
from socket import *
serverName = 'hostname'
serverPort = 12000
clientSocket = socket(AF_INET,  create UDP socket  
                                for server  
                                SOCK_DGRAM)
message = raw_input('Input lowercase sentence:')
clientSocket.sendto(message.encode(),  Attach server name,  
                                (serverName, serverPort)) port to message; send  
                                into socket
modifiedMessage, serverAddress =  read reply characters  
                                clientSocket.recvfrom(2048) from  
                                socket into string
print modifiedMessage.decode()  print out received string  
clientSocket.close() and close socket
```

Example: UDP server in Python

```
from socket import *
```

```
serverPort = 12000
```

```
serverSocket = socket(AF_INET,  create UDP socket  
                           SOCK_DGRAM)
```

```
serverSocket.bind(('', serverPort))  bind socket to local  
                                   port number 12000
```

```
print "The server is ready to receive"
```

```
While True:
```

```
    message, clientAddress = serverSocket.recvfrom(2048)  Read from UDP socket  
                                                         into message, getting  
                                                         client's address (client  
                                                         IP and port)
```

```
    modifiedMessage = message.decode().upper()
```

```
    serverSocket.sendto(modifiedMessage.encode(),  
                        clientAddress)
```

 send upper case string
back to this client