An Introduction to Python

Parham Alvani
Amirkabir University of Technology

parham.alvani@gmail.com
May 15, 2015

Keyboard Input

- The input() function reads a line from sys.stdin and returns it with the trailing newline stripped.

```
name = input("Enter your input: ")
print("Received input is : ", name)
```

File IO

- The open Function
- The file Object
- The close() Method
- The write() Method
- The read() Method
- See here for more details and functions.

The open Function

- Before you can read or write a file,
  you have to open it using Python's built-in open() function.
- This function creates a file object, which would be utilized
  to call other support methods associated with it.

```
file object = open(file_name [, access_mode][, buffering])
```

```
fo = open("foo.txt", "w")
```

The write() Method

- The write() method writes any string to an open file.

```
fileObject.write(string)
```

```
# Open a file
fo = open("foo.txt", "wb")
fo.write( "Python is a great language.\nYeah its great!!\n")

# Close opend file
fo.close()
```

The read() Method

- The read() method reads a string from an open file.

```
fileObject.read([count])
```

```
# Open a file
fo = open("foo.txt", "r+")
str = fo.read(10);
print "Read String is : ", str
# Close opend file
fo.close()
```

The readline() Method

- Read one entire line from the file.
- A trailing newline character is kept in the string.
- If the size argument is present and non-negative,
  it is a maximum byte count (including the trailing newline)
  and an incomplete line may be returned.

```
fileObject.readline([size])
```

The readlines() Method

- Read until EOF using readline() and return a list containing the lines thus read.

```
fileObject.readlines([sizehint])
```

Iterators & Generators

- There are many types of objects which can be used with a for loop. These are called iterable objects.
- The built-in function iter takes an iterable object and returns an iterator.

```
>>> x = iter([1, 2, 3])
>>> x
<listiterator object at 0x1004ca850>
>>> x.next()
1
>>> x.next()
2
>>> x.next()
3
```

Iterators & Generators

- Iterators are implemented as classes.

```
class yrange:
    def __init__(self, n):
        self.i = 0
        self.n = n

    def __iter__(self):
        return self

    def next(self):
        if self.i < self.n:
            i = self.i
            self.i += 1
            return i
        else:
            raise StopIteration()
```

Iterators & Generators

- In the above case, both the iterable and iterator are the same object. Noti

```
class zrange:
    def __init__(self, n):
        self.n = n

    def __iter__(self):
        return zrange_iter(self.n)
```

Iterators & Generators

```
class zrange_iter:
    def __init__(self, n):
        self.i = 0
        self.n = n

      def __iter__(self):
        # Iterators are iterables too.
        # Adding this functions to make them so.
        return self

    def __next__(self):
        if self.i < self.n:
            i = self.i
            self.i += 1
            return i
        else:
            raise StopIteration()
```

If both iteratable and iterator are the same object, it is consumed in a single iteration.

Modules

- A module allows you to logically organize your Python code.
- Grouping related code into a module makes the code easier to understand

Modules

```
def print_func( par ):
    print "Hello : ", par
    return
```

```
# Import module support
import support

# Now you can call defined function that module as follows
support.print_func("Zara")
```

Modules

- The current directory.
- Python then searches each directory in the shell variable PYTHONPATH.
- If all else fails, Python checks the default path.
  On UNIX, this default path is normally /usr/local/lib/python/.

Socket Programming

- Python provides two levels of access to network services
- To create a socket, you must use the socket.socket()
- socket_family: This is either AF_UNIX or AF_INET.
- socket_type: This is either SOCK_STREAM or SOCK_DGRAM.
- protocol: This is usually left out, defaulting to 0.

```
s = socket.socket (socket_family, socket_type, protocol=0)
```

Socket Programming

```
import socket                # Import socket module

s = socket.socket()          # Create a socket object
host = socket.gethostname() # Get local machine name
port = 12345                 # Reserve a port for your service.
s.bind((host, port))         # Bind to the port

s.listen(5)                  # Now wait for client connection.
while True:
    c, addr = s.accept()     # Establish connection with client.
    print('Got connection from', addr)
    c.send('Thank you for connecting')
    c.close()                # Close the connection
```

Socket Programming

```
import socket                   # Import socket module

s = socket.socket()             # Create a socket object
host = socket.gethostname() # Get local machine name
port = 12345                    # Reserve a port for your service.

s.connect((host, port))
print s.recv(1024)
s.close                         # Close the socket when done
```

Multithreading

```
class myThread (threading.Thread):
    def __init__(self, threadID, name, counter):
        threading.Thread.__init__(self)
        self.threadID = threadID
        self.name = name
        self.counter = counter
    def run(self):
        print "Starting " + self.name
        print_time(self.name, self.counter, 5)
        print "Exiting " + self.name

def print_time(threadName, delay, counter):
    while counter:
        time.sleep(delay)
        print "%s: %s" % (threadName, time.ctime(time.time()))
        counter -= 1
```

Multithreading

```
# Create new threads
thread1 = myThread(1, "Thread-1", 1)
thread2 = myThread(2, "Thread-2", 2)

# Start new Threads
thread1.start()
thread2.start()

print "Exiting Main Thread"
```

Questions?