Lesson Plan 7 | Form 2 | Using Network Sockets in Python

Objective

Students will be introduced to network sockets in Python, and by the end of the lesson, have a simple program capable of communicating with a central listener.

Warm-up

None.

Presentation (requires local wireless network)

Prior to class, place the listener Python program (see below for code) on the central server and run it.

Introduce the term “socket” to students, comparing it to an electrical outlet. Emphasize the bidirectionality of sockets: one can send information through a socket, and get information from the very same socket. Explain that sockets can be created and used in many programming languages, including Python.

Guided Practice

Have the students start their computers, go to their desktop, and create an empty Python script in gedit. Then have them open a terminal window, and place the two side by side.

Walk the students through each section of the client Python program (see below for code). Have students run their program after each section is added to see how things progress. Once everything has been added, have students send messages to the server and figure out the pattern it uses for its responses (the last number of the IP address is added to whichever number they send).

If time allows, ask students for ideas about how to get rid of the infinite loop to replace it with something more user-friendly, etc.

Independent Practice

None.

Closing

Have students save their program to their central file directory for additions, changes, etc. in later lessons.

In Hindsight

01/31/2017: Students seemed to enjoy this lesson, as it was not a lot of code but resulted in some interesting and new behavior. Some students went above and beyond to try different types of input,

including long messages, which caused buffer overruns and unexpected behavior since the server only retrieves 1024 bytes at a time (and thus the ends of long messages will bleed into the next message).

02/01/2017: The second stream also seemed to enjoy this lesson. One thing I found beneficial was to frequently stop and let them run their programs, even if it resulted in an error. This gave me an opportunity to ask students why we got the error, and what we could do to resolve it. I think this kind of practice will be beneficial for them when they do their own problem-solving on their own or without a teacher around.

Server Program

```
import socket, sys
from thread import start_new_thread
HOST = ''
PORT = 4002
MAX_CONNS = 100
try:
  s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
  s.bind((HOST, PORT))
  s.listen(MAX_CONNS)
  print "Socket created successfully on port: %d, now listening..." % PORT
except Exception:
  print "Error creating socket."
  sys.exit(1)
def client_thread(conn):
  print "Handling connection from %s:%s" % (addr[0], addr[1])
  conn.sendall("Hello " + addr[0] + ", you are now talking to a Python program on the server.")
  while True:
    data = conn.recv(1024)
    if not data:
      break
    try:
      user_num = int(data)
      user_num_plus_last_ip_byte = user_num + int(addr[0].split('.')[3])
      conn.sendall("You sent me " + str(user_num) + \
                   ", and I give you " + str(user_num_plus_last_ip_byte) + " back.")
    except:
      conn.sendall("You didn't send me an integer, try again.")
  conn.close()
  print "Closed connection from %s:%s" % (addr[0], addr[1])
try:
  while True:
    # blocking call
    conn, addr = s.accept()
```

```
    start_new_thread(client_thread, (conn,))
except KeyboardInterrupt:
  print "Interrupt received, closing server."
  s.close()
```

Client Program

```
import socket

IP = ‘10.0.2.100’
PORT = 4002

s = socket.socket()
s.connect((IP, PORT))
print s.recv(256)

while True:
  to_send = raw_input(“Enter a number to send to the server: “)
  s.send(str(to_send))
  print s.recv(256)

s.close()
```