

Introducción a Python y Sockets

Redes y Sistemas Distribuidos

Juan A. Fraire



Introducción a Python

(el lenguaje del Lab 1 y 2)

Introducción

Bucles y Condicionales

Listas y Diccionarios

Operadores y Mutabilidad

Avanzado

PEP8

Debugging



Introducción

- Es **interpretado**
- Tiene **tipado dinámico**
- Tiene **garbage collector**
- Usa **indentación** para bloques (no {})
- Ejecución
 - **Interactiva** `$python3` luego consola `>>>`
 - **Archivo** `$python3 file.py`
- Versiones
 - **Python 2.x**: legacy
 - **Python 3.x**: future <- vamos por esta

```
# dynamic type:
```

```
a = 10
```

```
a = " algo "
```

```
# indentation: this is one thing
```

```
if a == "algo":
```

```
    print (a) # python 3
```

```
a = "algo"
```

```
# indentation: this is another thing
```

```
if a == "algo":
```

```
    print (a)
```

```
    a = "algo"
```



Bucle y Condicional

Lo muy básico

```
x = 10 + 2      # Asignación y suma
x = "mo" + "no" # Concatenacion
x = [1, "1"]    # Listas , diversos tipos
x = (2, "a", 1.1) # Tuplas
x = sqrt(2)     # Llamado a funcion
```

Asignación

```
a = 10          # Asignación int
b = "hello"     # Asignación string
a,b = b,a       # Asignación múltiple
a = "hello"
b = 10
```

Bucle

```
xs = []          # Nueva lista vacía
while len(xs) < 10: # Mientras tamaño<10
    xs.append(len(xs)) # Agregar tamaño xs
print (xs)
```

Condicional

```
xs = ["cadena"] # Lista con un elemento
if len(xs) == 0: # Tamaño de xs
    xs.append(3) # Agregar elemento 3
else:
    xs.append("s") # Agregar elemento "s"
print (xs)        # out: ["cadena","s"]
```



Listas y Diccionarios

Definición de funciones

```
def rango(i, j):  
    xs = []  
    while len(xs) < j-i:  
        xs.append(len(xs) + i)  
    return xs  
print (rango(10, 15)) # [10, 11, 12, 13, 14]
```

For itera sobre elementos en una lista

```
n = 0  
for x in rango (1, 10):  
    n += x  
print (n) # 45
```

Listas, strings y diccionarios

```
ls = range(0, 10)  
s = "abcdefghij"  
print ls[0] # 0  
print s[-1] # "j"  
print ls[1:9] # [1, 2, 3, 4, 5, 6, 7, 8]  
print s[:-2] # 'abcdefgh'  
print ls[2:] # [2, 3, 4, 5, 6, 7, 8, 9]
```

```
d = {} # Un diccionario vacío  
d[1] = "string" # key: int  
d["string"] = 2 # key: string  
d[(1, 7, "cosa")] = 0 # key: tupla  
print d["string"] # 2  
print d[d[1]] # 2  
print d # {1:'string', 'string':0}
```



Operadores y Mutabilidad

```
# Los booleanos: True y False
# Los operadores: and, or, not, in
d = [1, 3, 5]
print (2 in d) or (4 not in d)
```

```
# Import de librerías
```

```
import sys
print (sys.Argv)

from os import environ
print (environ["USER"])
```

```
# Números, string y tuplas son inmutables
```

```
a = "hello"
b = a                # b = "hello"
a = a + "world"
print (a)            # a = "hello world"
print (b)            # b = "hello"
```

```
# Listas, diccionarios y clases son mutables
```

```
a = ["hello"]
b = a
a.append("world")
print (a)            # ['hello', 'world']
print (b)            # ['hello', 'world']
```



Avanzado

- Declaración de objetos abstractos

```
# Abstract objects (classes)
class MyClass:
    i = 12345
    def f(self):
        return 'hello world'
```

- Iterables por **comprensión**
- **Sobreescritura** de métodos y **polimorfismo**
- Manejos de **errores** (try/except)

```
# Lists by comprehension
S = [x for x in range(101) if x % 3 == 0]
```

```
# Error handling
try:
    print "Hello World"
except IOError:
    print('An error trying to read file.')
except ValueError:
    print('Non-numeric data found.')
except:
    print "This is an error message!"
```



PEP8

- El código es leído muchas más veces de lo que es escrito
- **Convenciones de escritura**
 - Los docentes las evaluamos y las agradecemos!
- PEP8
 - Indentación: usa 4 (cuatro) **espacios** por indentación, el uso de **tabs** no está recomendado
 - Máximo largo de línea: 79 caracteres
 - Líneas en blanco
 - Separa definiciones de **clase** con 2 (dos) líneas en blanco
 - Separa **funciones** dentro de la clase con 1 (una) línea en blanco
 - Separa **unidades lógicas** dentro de funciones con 1 (una) línea en blanco
 - Revisar <https://pep8.org>



PEP8 - Indentación

Si

```
# Aligned with opening delimiter.
foo = long_function_name(var_one, var_two,
                          var_three, var_four)

# More indentation included to distinguish
# this from the rest.
def long_function_name(
    var_one, var_two, var_three,
    var_four):
    print(var_one)
```

No

```
# Arguments on first line forbidden when not
# using vertical alignment.
foo = long_function_name(var_one, var_two,
                          var_three, var_four)

# Further indentation required as indentation is
# not distinguishable.
def long_function_name(
    var_one, var_two, var_three,
    var_four):
    print(var_one)
```



PEP8 - Espacios

Si

```
# Avoid extraneous whitespace:
# Immediately inside parentheses, brackets or
braces:
spam(ham[1], {eggs:2})

# Between a trailing comma and a following
close parenthesis:
foo = (0, )

# Immediately before a comma, semicolon, or
colon:
if x == 4: print x, y; x, y = y, x

# Immediately before a parenthesis with args.
spam(1)
```

No

```
# Avoid extraneous whitespace:
# Immediately inside parentheses, brackets or
braces:
spam( ham[ 1 ], { eggs: 2 } )

# Between a trailing comma and a following close
parenthesis:
bar = (0, )

# Immediately before a comma, semicolon, or
colon:
if x == 4 : print x , y ; x , y = y , x

# Immediately before a parenthesis with args.
spam (1)
```



Debugging - Print

- **Ventajas**

- Rápido para encontrar errores en programas pequeños

- **Desventajas**

- Difícil de controlar en bucles
- Sólo variables numéricas o strings
- Podemos olvidarnos de quitar prints
- Print comentados generan un mal aspecto

```
import sys

# Example
def main():
    texto = ''
    for palabra in sys.argv[1:]:
        texto = texto + ' ' + palabra
    print texto
```



Debugging - Depuradores

- GDB, IPDB, PDB

- **Ventajas**

- Puntos de interrupción en el código para revisar valor de variables
- Estudio del código línea por línea, función por función, y estudiar la evolución de las variables

- **Desventajas**

- Tedioso de aprender

- **Enlaces**

- <https://pypi.python.org/pypi/ipdb>

IPDB

```
$ sudo pip install ipdb  
$ ipdb example.py
```

```
# example.py  
import ipdb  
def main():  
    texto = ''  
    for palabra in sys.argv[1:]:  
        texto += ' ' + palabra  
    ipdb.set_trace()  
    print texto
```

```
ipdb> s(tep)  
ipdb> c(ontinue)
```



Introducción a Sockets

(la API del Lab 1 y 2)

Introducción

Puertos

Conexiones

Transporte de Datos

Sockets



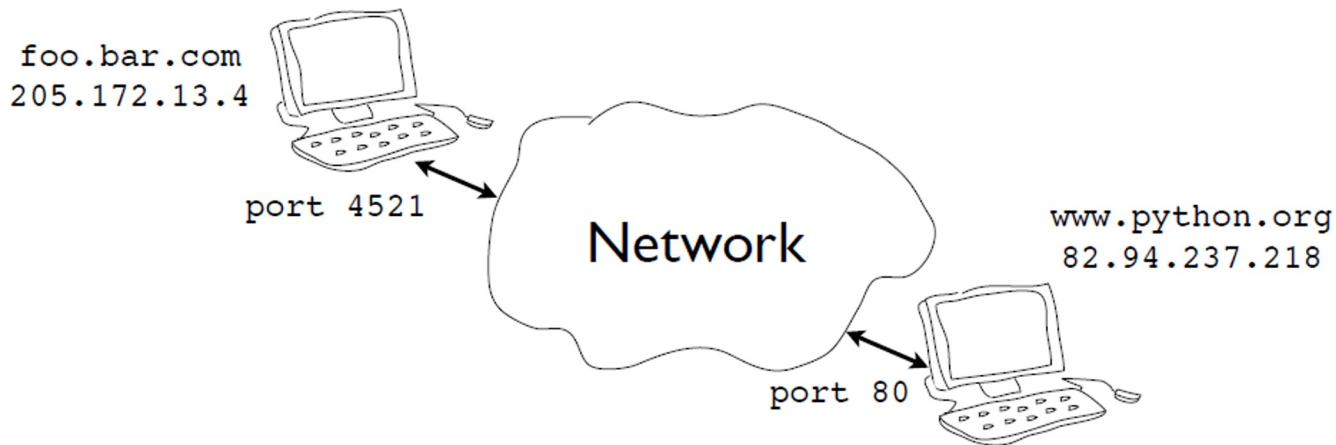
Introducción

- **Programar para redes**, uno de los usos más populares de Python



Introducción - Cliente/Servidor

- Un **cliente** envía y recibe bits de un **servidor** por medio de una **red**
 - **Direccionamiento** (URL/IP para nodo destino y **puerto** para el servicio destino)
 - **Transporte de datos** (TCP servicio stream y **UDP** servicio datagrama)



Puertos

- Puertos para **servicios conocidos** son reservados y fijos (**0 a 1023**):
 - 21 **FTP**
 - 22 **SSH**
 - 23 **Telnet**
 - 25 **SMTP** (Mail)
 - 80 **HTTP** (Web)
 - 110 **POP3** (Mail)
 - 443 **HTTPS** (web)
- Puertos **registrados** pueden reservarse (**1024 a 49151**)
- Puertos **dinámicos privados** que no pueden reservarse (**49152 a 65535**)



Puertos - Comando netstat

```
shell % netstat -a
```

```
Active Internet connections (servers and established)
```

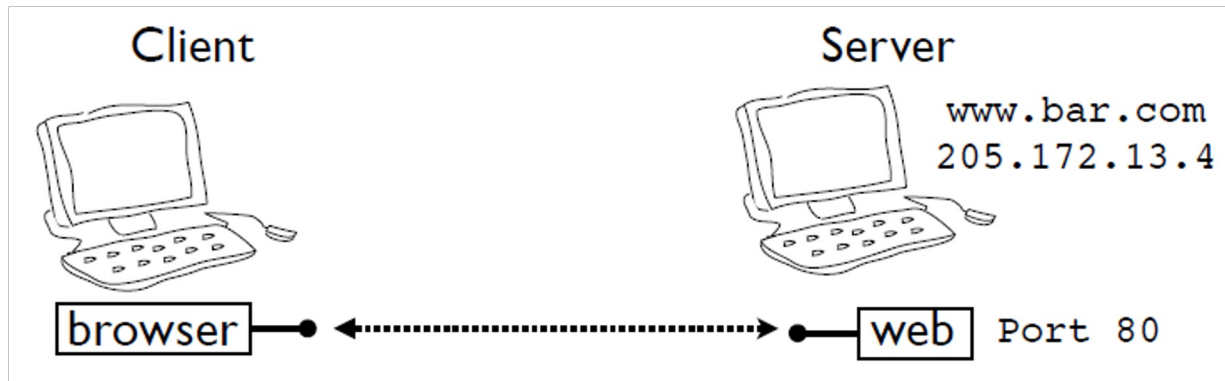
Proto	Recv-Q	Send-Q	Local Address	Foreign Address	State
tcp	0	0	*:imaps	*.*	LISTEN
tcp	0	0	*:pop3s	*.*	LISTEN
tcp	0	0	localhost:mysql	*.*	LISTEN
tcp	0	0	*:pop3	*.*	LISTEN
tcp	0	0	*:imap2	*.*	LISTEN
tcp	0	0	*:8880	*.*	LISTEN
tcp	0	0	*:www	*.*	LISTEN
tcp	0	0	192.168.119.139:domain	*.*	LISTEN
tcp	0	0	localhost:domain	*.*	LISTEN
tcp	0	0	*:ssh	*.*	LISTEN
...					



Conexiones

- Un **servidor** corre una aplicación
 - **Identificador de host** y un **puerto**
 - En Python se expresa como una **tupla**
- Un **cliente** inicia una conexión

```
# Server host and port tuple  
("www.python.org", 80)  
("205.172.13.4", 443)
```



Conexiones - Request/Response

- La mayoría de los intercambios son del tipo **request/response**
- Un **cliente** envía un mensaje **request** (HTTP)
 - `GET /index.html HTTP/1.0`
- El **servidor** responde con un mensaje **response** (HTTP)
 - `HTTP/1.0 200 OK`
`Content-type: text/html`
`Content-length: 48823`
`<HTML>`
`...`
- El formato depende del protocolo de aplicación



Conexiones - Request/Response

- El servicio **telnet** puede usarse para comunicarse con servidores
 - **shell% telnet www.python.org 80**
 - Trying 82.94.237.218...
Connected to www.python.org.
Escape character is '^]'.
GET /index.html HTTP/1.0
 - HTTP/1.1 200 OK
Date: Mon, 31 Mar 2008 13:34:03 GMT
Server: Apache/2.2.3 (Debian) DAV/2 SVN/1.4.2
mod_ssl/2.2.3 OpenSSL/0.9.8c
...



Transporte de Datos

- **Stream (TCP)**
 - Se establece una conexión y se leen/escriben datos en un flujo continuo de bytes
 - Similar a escribir un archivo
- **Datagram (UDP)**
 - Se envían paquetes discretos (o mensajes)
 - Cada paquete contiene una colección de bytes independiente y autónomo

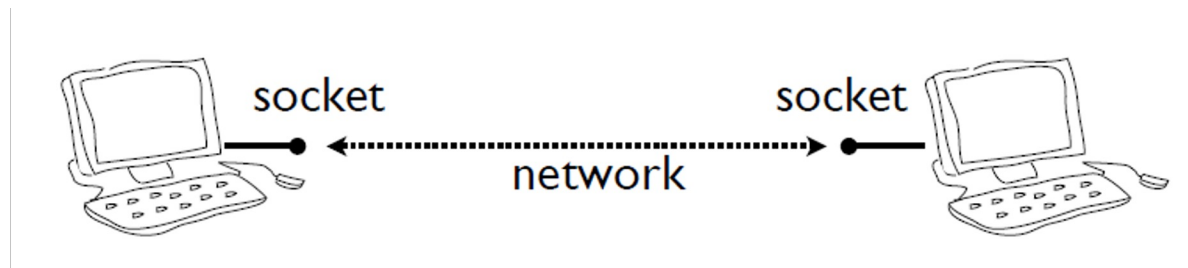


Socket

- Abstracción para programación en red
 - Endpoint para conexiones en **ambas direcciones**

entonces tengo que aclarar si es

uno o dos cuando creo el socket



Socket - Creación

```
# Create socket
import socket
s = socket.socket(addr_family, type)

# addr_family:
# -socket.AF_INET      Internet protocol (IPv4)
# -socket.AF_INET6     Internet protocol (IPv6)
# type:
# -socket.SOCK_STREAM  Stream (TCP)
# -socket.SOCK_DGRAM   Datagrams (UDP)

from socket import *
s = socket(AF_INET, SOCK_STREAM)
```

Una vez creado, el uso del socket dependerá de la aplicación:

- **El cliente** iniciará una conexión saliente
- **El servidor** escuchará conexiones entrantes



Socket - Cliente TCP

```
# Create stream (TCP) IPv4 socket
from socket import *
s = socket(AF_INET, SOCK_STREAM)

# Connect to host at port 80 (http)
s.connect(("www.python.org", 80))

# Send byte stream
s.send(b"GET /index.html HTTP/1.0\n\n")

# Receive byte data
data = s.recv(10000) # Get response
s.close()
```

- **s.connect** inicia una conexión con un host en un puerto
- **s.send** transmite datos
 - Retorna bytes transmitidos
- **s.recv** recibe datos
 - El argumento **10000** indica máximos bytes a recibir
- **s.close** cierra el socket



Socket - Servidor TCP (1 conexión)

```
# Create stream (TCP) IPv4 socket
from socket import *
s = socket(AF_INET, SOCK_STREAM)

# Bind and listen in port 9000
s.bind(("", 9000)) # empty string = localhost
s.listen(5)

while True:
    c, a = s.accept()
    print ("Connection from %s\n" % a[0])
    c.send(b"Hello %s\n" % a[0])
    c.close()
```

- **s.bind** indica la dirección local
- **s.listen** inicia escucha
- **s.accept** acepta nueva conexión
 - c - nuevo socket para conexión
 - a - ("104.23.11.4", 27743)
- Con **telnet**
 - % telnet localhost 9000
Connected to localhost.
Escape character is '^]'.
Hello 127.0.0.1
Connection closed by foreign host.



Socket - Servidor TCP (1 conexión)

```
# Client
```

```
...
```

```
>>> len(data)
```

```
1000000
```

```
>>> s.send(data)
```

```
37722
```

```
>>> s.send(moredata)
```

```
...
```

```
# Server
```

```
...
```

```
>>> data = s.recv(maxsize)
```

```
>>> len(data)
```

```
6420
```

```
...
```

Escrituras y lecturas parciales

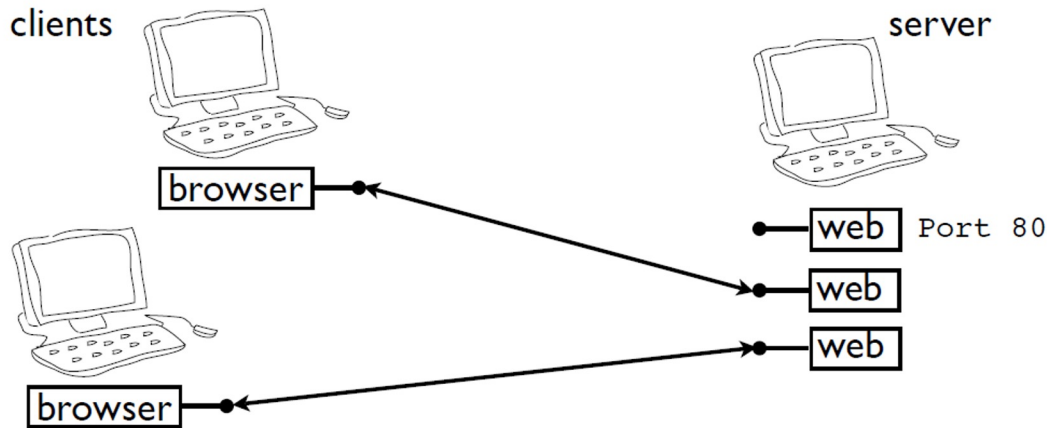
En TCP, la transmisión de datos es continuo (**stream**):

- `recv(maxsize)` puede retornar datos de los `send()` combinados, o parte del primero
- Dependerá del sistema operativo, ancho de banda de la red, congestión, etc
- **`s.sendall`** bloquea hasta transmitir la totalidad
- **`s.recv`** devuelve un string vacío '' cuando no hay más datos



Socket - Servidor TCP (N conexiones)

- Usualmente, los servidores manejan **múltiples conexiones**
- Cada cliente tiene su socket en el servidor



```
while True:
    c,a = s.accept()

# c is a socket
# connecting the server
# with the accepted
# client
```

- Threads
- Forks
- Async Server

