



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



Introducción a la programación de sockets con Python.

**Prácticas de laboratorio
de Diseño de Servicios Telemáticos**

OBJETIVOS.

El objetivo de la práctica es comprender como podemos programar fácilmente aplicaciones de red usando la librería Socket disponible para Python.

Para ello vamos a realizar una serie de ejercicios utilizando tanto el protocolo de transporte UDP como el protocolo TCP.

1. Introducción

Una aplicación de red consta de parejas de procesos que se ejecutan en equipos terminales distintos o incluso en el mismo equipo terminal que se comunican, intercambiándose mensajes. En este contexto, el proceso que inicia la comunicación se designa como **cliente** mientras que el proceso que espera a ser contactado para comenzar la sesión de comunicación se denomina **servidor**. Por ejemplo, en la aplicación Web un proceso navegador es un proceso cliente y el servidor Web es un proceso servidor.

Los mensajes enviados desde un proceso a otro deben atravesar la red subyacente (ver Figura 1). Así en la arquitectura TCP/IP se utiliza un interfaz software denominado **socket** que no es más que el interfaz entre la capa de aplicación y la capa de transporte¹.

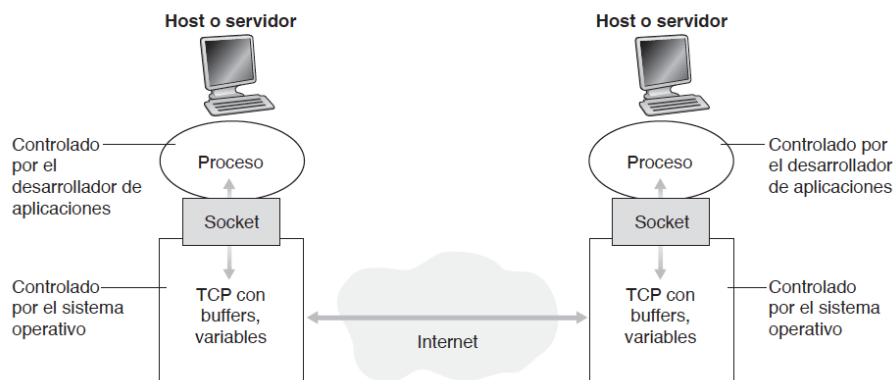


Figura 1. Procesos de aplicación, sockets y protocolo de transporte subyacente.

Por lo tanto, desde el punto de vista de una aplicación, un socket representa “un canal de comunicación” entre los dos procesos quedando definido por: un par de direcciones IP (local y remota), un protocolo de transporte y un par de números de puerto (local y remoto).

Desde un punto de vista práctico, los sockets pueden ser de dos tipos en función del protocolo de transporte empleado (UDP o TCP). En esta práctica vamos a introducir la programación de ambos tipos de sockets.

¹ Para más información consulte los apartados 2.1 y 2.7 del libro Redes de computadoras: Un enfoque descendente (7ª Edición) de James Kurose y Keith Ross.

2. Desarrollo inicial de un Cliente y un Servidor con UDP

En este apartado vamos a desarrollar un cliente y un servidor que utilizan como protocolo de transporte UDP.

Cliente.

Utilizando un editor de código, por ejemplo, VSCode, cree en la carpeta deseada un archivo Python con nombre `UDP_Client.py` con el siguiente código:

```
import socket
UDP_address = "127.0.0.1"
UDP_port = 15002
Message = input('Introduzca su mensaje ')
clientSock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
clientSock.sendto(Message.encode(), (UDP_address, UDP_port))
clientSock.close()
```

La explicación del código del programa `UDP_Client.py` es bien sencilla. En la línea 1 importamos el módulo `socket`. Incluyendo esta línea, podemos crear sockets dentro de nuestro programa. Creamos un socket denominado `clientSock` donde el segundo parámetro permite especificar el tipo de socket que se va a emplear. Dado que vamos a programar un cliente UDP el socket es de tipo `socket.SOCK_DGRAM`.

Utilizando el socket creado anteriormente (`clientSock`) enviamos el mensaje a la dirección IP y puerto destino utilizando el método `".sendto()"`. Fijaros que como el protocolo UDP no es orientado a conexión, el método `".sendto()"` crea un paquete al que le asocia la dirección IP y puerto destino que le aportamos. Cabe destacar que por el socket sólo se pueden enviar bytes por lo que el método `".encode()"` se encarga de convertir el mensaje.

Servidor

De la misma forma creamos un segundo archivo denominado `UDP_Server.py` con el siguiente código:

```
import socket
UDP_address = "127.0.0.1"
UDP_port = 15002
serverSock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
serverSock.bind((UDP_address, UDP_port))
print("el Servidor está listo")
message, clientAddress = serverSock.recvfrom(2048)
print(message.decode())
serverSock.close()
```

Estudiando el código podemos observar que es similar al anterior. La principal diferencia es que se utiliza el método `".bind"` que asigna la dirección IP y número de

puerto ("127.0.0.1" y 15002) al socket. De este modo, cuando alguien envía un paquete al puerto 15002 en la dirección IP del Servidor, dicho paquete será dirigido a este socket.

El servidor recibe el mensaje que llega al socket mediante el método `".recvfrom()"`. Cómo salida de esta función se obtienen dos valores: los datos del mensaje (que se almacenan en la variable `message`), y la dirección de origen del paquete (que se almacena en la variable `clientAddress`). El método `".recvfrom()"` también especifica el tamaño de buffer de 2048 como entrada. Tamaño suficiente para la mayor parte de aplicaciones. Recordad que por el socket se reciben bytes por lo que para convertir el mensaje a caracteres se utiliza el método `".decode()"`.

Para comprobar el funcionamiento de forma muy pedagógica abrimos dos ventanas de comandos. Nos movemos hasta la carpeta donde se encuentran los 2 códigos y ejecutamos cada uno en una ventana, por ejemplo `c:\py>py UDP_Client.py`

Piense si es necesario ejecútalos en algún orden en concreto...

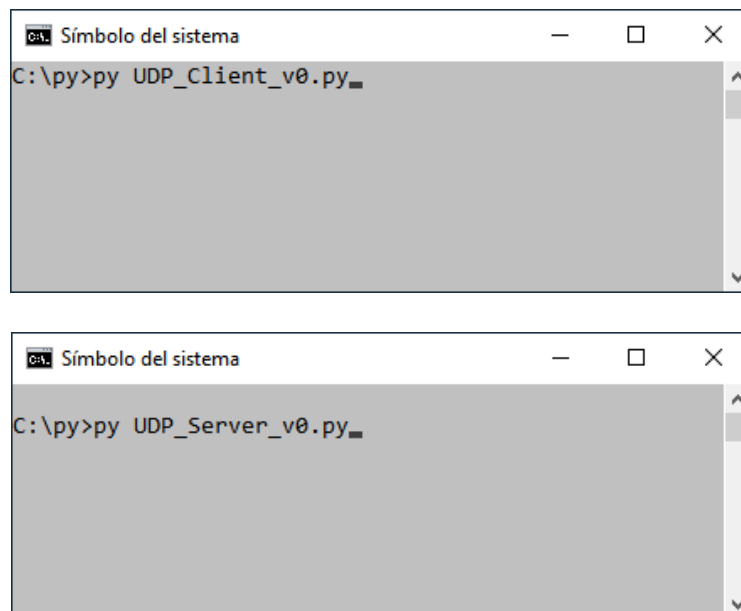


Figura 2. Ejecución de los programas utilizando dos ventanas de comando.

3. Desarrollo de un Cliente y un Servidor UDP modificados

Vamos a mejorar los códigos anteriores. Queremos que el Servidor también envíe datos al Cliente, es decir que la comunicación sea bidireccional, y que los sockets no se cierren nada más enviar o recibir un único mensaje.

Cliente

Modificamos el código Cliente, cambiando el nombre (por ej. UDP_client_v2.py) para no perder el código anterior.

```
import socket
UDP_address = "127.0.0.1"
UDP_port = 16000
clientSock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

while True:
    Message = input ('Introduzca su mensaje en minúsculas: ')
    clientSock.sendto(Message.encode(), (UDP_address, UDP_port))
    modifiedMessage, address_received = clientSock.recvfrom(2048)
    print(modifiedMessage.decode())
    if (modifiedMessage.decode()=="FIN"): break
clientSock.close()
```

Servidor

De la misma forma escribimos el nuevo Servidor UDP_server_v2.py:

```
import socket
UDP_address = "127.0.0.1"
UDP_port = 16000
serverSock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
serverSock.bind((UDP_address, UDP_port))
print("el Servidor está listo")

while True:
    message, clientAddress = serverSock.recvfrom(2048)
    print(message.decode())
    modifiedMessage = message.decode().upper()
    serverSock.sendto(modifiedMessage.encode(), clientAddress)
    if (modifiedMessage=="FIN"): break
serverSock.close()
```

Desarrollo de un sencillo chat UDP

Utilizando el código anterior no debería ser complicado modificarlo para realizar un sencillo chat entre el Cliente y el Servidor.

Vamos a conformarnos con que los mensajes se envíen de forma alterna, empezando por el cliente. Es decir:

- El Cliente envía el mensaje que teclea el usuario y espera la recepción de la respuesta que le enviará el Servidor.
- El Servidor espera un primer mensaje del Cliente y después envía la respuesta tecleada por el usuario del Servidor.
- El proceso se repite hasta que cualquiera de los dos extremos introduzca el texto "fin". Cuando esto ocurre los sockets de los dos extremos tienen que cerrarse.

Tarea: Realice el código del chat cliente y del chat servidor.

4. Desarrollo de un Cliente y un Servidor con TCP

A continuación se muestra el código básico de un Servidor y un Cliente que utilizan sockets TCP. Cree dos archivos: TCP_Server.py y TCP_Client.py.

Servidor

Observando el código del Servidor podemos encontrar las siguientes diferencias respecto al Servidor basado en UDP:

- A la hora de crear el socket tenemos que indicar que el socket es de tipo TCP. Por lo que se indica que el tipo es SOCK_STREAM. Por lo que la sentencia que se utilizará será la siguiente:

```
serverSock = socket(socket.AF_INET, SOCK_STREAM).
```
- Se utiliza el método ".listen()" para poner el socket en estado de escucha. El entero que se pasa como parámetro es la longitud máxima de la cola de conexiones pendientes.
- Se utiliza el método ".accept()" para aceptar conexiones entrantes al Servidor. El resultado es un objeto de tipo socket, que se empleará para enviar y recibir datos, y la dirección IP del Cliente que ha realizado la conexión.
- Se utiliza el método ".send()" para enviar el paquete a través del socket. En este caso, como el protocolo TCP es orientado a conexión, y la conexión ya está establecida, no es necesario indicar la dirección IP y puerto destino.

```
import socket
serverAddress = "127.0.0.1"
serverPort = 19000
serverSock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
serverSock.bind((serverAddress, serverPort))
print("el Servidor está listo")
serverSock.listen(5)

while True:
    clientSock, addr = serverSock.accept()
    print(f"Conexión desde {addr} establecida!")
    clientSock.send(bytes("Bienvenido al Servidor", "utf-8"))
    clientSock.close()
```

Cliente

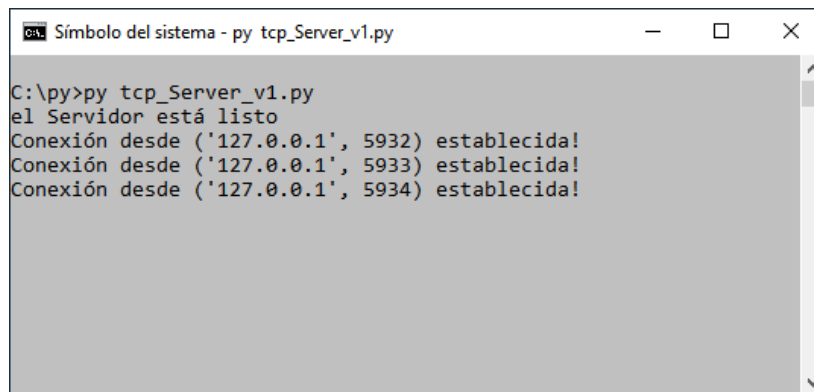
A continuación, se muestra el código básico de un Cliente TCP. La principal diferencia con el Cliente UDP es que ahora, antes de que el Cliente pueda enviar datos al Servidor (o viceversa), se debe establecer una conexión TCP. Para ello se utiliza el método “.connect()”.

Para recibir mensajes se utiliza el método “.recv()” en lugar de “.recvfrom()” ya que la conexión con el servidor ya está establecida.

```
import socket
serverAddress = "127.0.0.1"
serverPort = 19000
clientSock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
clientSock.connect((serverAddress, serverPort))

receivedMessage = clientSock.recv(1024)
print(receivedMessage.decode())
clientSock.close()
```

Ejecute los dos programas en dos ventanas de comandos. Observe como el Servidor queda abierto a la espera de conexiones. Explique por qué va cambiando el mensaje que aparece en la ventana del Servidor.



```
Símbolo del sistema - py tcp_Server_v1.py
C:\py>py tcp_Server_v1.py
el Servidor está listo
Conexión desde ('127.0.0.1', 5932) establecida!
Conexión desde ('127.0.0.1', 5933) establecida!
Conexión desde ('127.0.0.1', 5934) establecida!
```

Figura 3. Ejecución del programa TCP_Server.py.

5. Sistemas de autenticación de doble factor 2FA

En la actualidad, cuando queremos acceder a una aplicación o un sitio web, utilizamos sistemas denominados autenticación de doble factor, 2FA, que obligan a realizar dos verificaciones diferentes de la identidad de un usuario.

2FA nos da una capa adicional de seguridad. Incluso si alguien conoce la contraseña, no podrán acceder a la cuenta o sistema sin la segunda forma de autenticación. Las formas de autenticación suelen ser de diferentes categorías:

- **Algo que el usuario sabe:** Esto generalmente se refiere a una contraseña o PIN que solo el usuario debería conocer.
- **Algo que el usuario posee:** Esto podría ser un dispositivo móvil, una tarjeta, o incluso una aplicación de autenticación en el teléfono.
- **Algo que el usuario es:** Esta categoría implica características del usuario, como huellas dactilares, reconocimiento facial o voz.

Uno de los mecanismos de autenticación más populares actualmente son los denominados OTP, o "One-Time Password" basados en un código de acceso de uso único que es válido para un acceso y sólo por un corto período de tiempo. Los códigos se pueden enviar por correo electrónico, en un mensaje de texto SMS o ser generados en una aplicación de autenticación como Goggle Authenticator o Microsoft Authenticator. Estas aplicaciones tienen la ventaja de que pueden funcionar sin conexión a Internet, y generan el código de acceso mediante un algoritmo denominado TOTP "Time-based One-Time Password"

TOTP genera un código de un sólo uso a partir de una clave secreta y un número que se corresponde con el instante de tiempo (Unix Time) en que se genera el código. El proceso es el siguiente: Cuando un usuario se registra por primera vez el Servidor éste genera una clave secreta aleatoria que almacena de forma segura. La clave secreta se hace llegar al usuario, normalmente por medio de un código QR, que el usuario incorporará en su aplicación de autenticación instalada en su teléfono móvil. A partir de ese momento la aplicación puede generar un código de acceso cuando se le solicite,

calculando un valor TOTP a partir de la clave secreta y el valor del instante de tiempo. El servidor puede generar el mismo código realizando el mismo proceso, ya que tiene almacenada la clave secreta de todos los usuarios.

Vamos a comprobar todo esto. Lo primero es instalar una Aplicación de autenticación. En la UPV utilizamos la aplicación de Microsoft, por ejemplo, para poder realizar una VPN desde casa, así que ya debería estar instalada en el móvil. Es posible, sin embargo, utilizar cualquier otra, como la de Google.



Figura 4. Aplicaciones de Autenticación

Como hemos comentado un Servidor debería generar una clave secreta o password para cada usuario y almacenarla de forma segura. Nosotros vamos a utilizar el número de dni (únicamente los números) como clave secreta. Necesitamos un código QR correspondiente para introducirlo a la aplicación. Para ello, cree un archivo denominado `pyotp_uri.py`. Este código utiliza la librería `pyotp` que implementa el mecanismo TOTP (github.com/pyauth/pyotp).

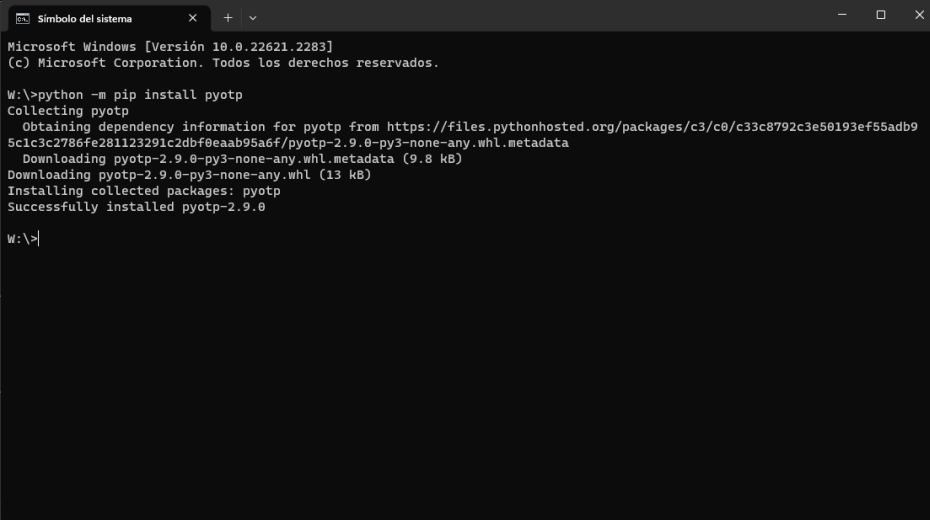
Para poder utilizar la librería en nuestras aplicaciones Python es necesario instalarla. Para ello (en función del sistema operativo que estéis trabajando) ejecuta uno de los siguientes comandos:

- Linux/macOS (ver Figura 5): Abra una ventana de Terminal y escriba el comando `python3 -m pip install pyotp`.

```
jomorros — -zsh — 80x24
Last login: Wed Sep 20 13:59:05 on ttys001
jomorros@iMac-2 ~ % python3 -m pip install pyotp
DEPRECATION: Configuring installation scheme with distutils config files is deprecated and will no longer work in the near future. If you are using a Homebrew or Linuxbrew Python, please see discussion at https://github.com/Homebrew/homebrew-core/issues/76621
Collecting pyotp
  Obtaining dependency information for pyotp from https://files.pythonhosted.org/packages/c3/c0/c33c8792c3e50193ef55adb95c1c3c2786fe281123291c2dbf0eaab95a6f/pyotp-2.9.0-py3-none-any.whl.metadata
  Downloading pyotp-2.9.0-py3-none-any.whl.metadata (9.8 kB)
Using cached pyotp-2.9.0-py3-none-any.whl (13 kB)
Installing collected packages: pyotp
  DEPRECATION: Configuring installation scheme with distutils config files is deprecated and will no longer work in the near future. If you are using a Homebrew or Linuxbrew Python, please see discussion at https://github.com/Homebrew/homebrew-core/issues/76621
  DEPRECATION: Configuring installation scheme with distutils config files is deprecated and will no longer work in the near future. If you are using a Homebrew or Linuxbrew Python, please see discussion at https://github.com/Homebrew/homebrew-core/issues/76621
Successfully installed pyotp-2.9.0
jomorros@iMac-2 ~ %
```

Figura 5. Instalación de una librería utilizando pip en Linux/macOS.

- Windows (ver Figura 6): Abra la línea de comandos ejecutando el programa `cmd.exe` y escriba el comando `python -m pip install pyotp`.



```
Símbolo del sistema
Microsoft Windows [Versión 10.0.22621.2283]
(c) Microsoft Corporation. Todos los derechos reservados.

W:\>python -m pip install pyotp
Collecting pyotp
  Obtaining dependency information for pyotp from https://files.pythonhosted.org/packages/c3/c8/c33c8792c3e50193ef55adb95c1c3c2786fe281123291c2dbf0eaab95a6f/pyotp-2.9.0-py3-none-any.whl.metadata
  Downloading pyotp-2.9.0-py3-none-any.whl.metadata (9.8 kB)
  Downloading pyotp-2.9.0-py3-none-any.whl (13 kB)
Installing collected packages: pyotp
Successfully installed pyotp-2.9.0

W:\>
```

Figura 6. Instalación de una librería Python utilizando pip en Windows.

El código a desarrollar se muestra a continuación. Será necesario cambiar el valor del dni, mientras que como nombre de usuario podemos poner el usuario de la UPV, o dejarlo como en el ejemplo, ya que no afecta para nada al funcionamiento o a la generación de códigos. Simplemente es el nombre que mostrará la App de autenticación.

La clave secreta (password) suele estar siempre codificada en base32, para evitar confusiones si se introdujera a mano. Por tanto, la variable secret es el valor codificado en base32 de nuestro dni.

```
import pyotp
import base64
dni = "123456"
secret= base64.b32encode(bytearray(dni, 'ascii')).decode('utf-8')
print("mi secreto:", secret)

totp_object = pyotp.TOTP(secret)
qr_text = totp_object.provisioning_uri(name="mi_usuario",
issuer_name="Mi App")
print (qr_text)
```

El resultado al ejecutar es el URI que debería llevar el código QR. Por ejemplo:
otpaauth://totp/Mi%20App:mi_usuario?secret=GEZDGNBVGY%3D%3D%3D%3D%3D%3D&issuer=Mt%20App

Para generar la imagen QR podemos utilizar la web the-qrcode-generator.com. y pegar el URI que se ha generado con el código python. Observad que hay que hacerlo en la pestaña "Plain Text".

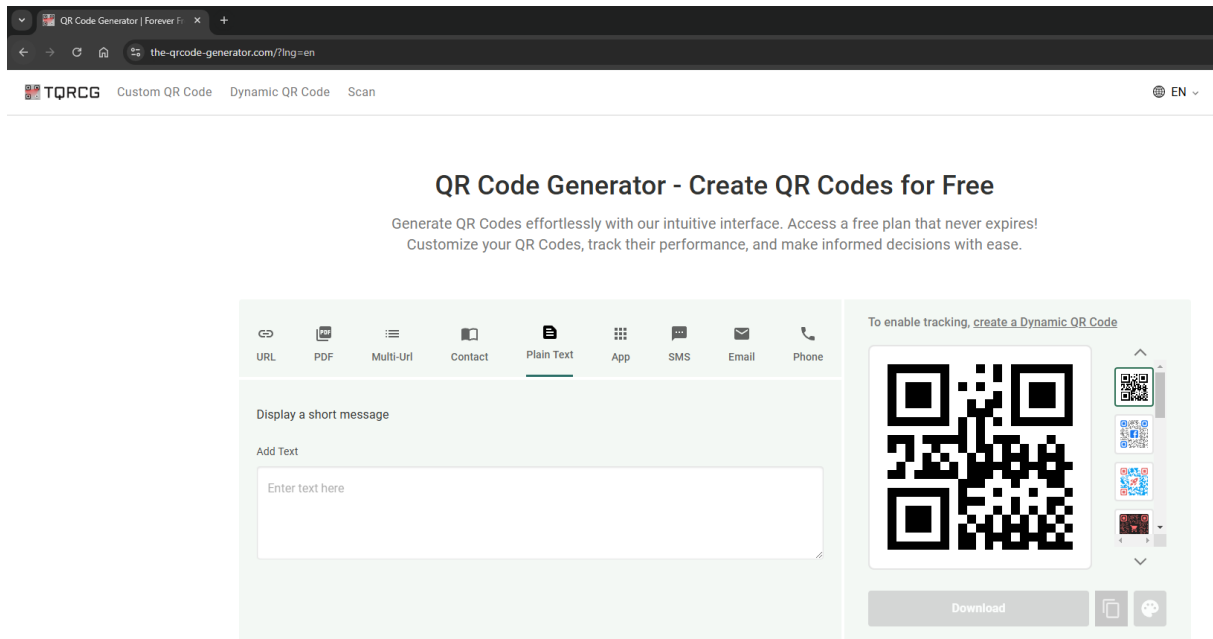


Figura 7. Generación código QR.

Agregue una nueva cuenta en la aplicación de Autenticación. Con Microsoft Authenticator se corresponde con “Cuenta Personal”, y simplemente hay que mostrarle la imagen del código QR. A partir de ese momento la aplicación de autenticación genera códigos de un uso basados en TOTP.

Para comprobar que todo funciona añada las siguientes líneas al código anterior.

```
otp = input("ingresa el código OTP:")
valid = totp_object.verify(otp)
print(valid)
```

Ejecute el código. El código programado nos solicita el código OTP. Compruebe que el código generado por nuestra App del móvil es un código válido.

6. Desarrollo de un cliente

Para finalizar la práctica, y como mecanismo para confirmar que se ha realizado por parte del alumno, el alumno tiene que programar un **Cliente TCP**. Este cliente se debe conectar con un Servidor ya implementado por los profesores de la asignatura y que se está ejecutando en un PC del laboratorio. La dirección IP del Servidor es **158.42.32.220** y el puerto TCP abierto es el **21000**. Si está realizando la práctica desde fuera del campus tendrá que realizar primero una conexión VPN para obtener una IP de la Universidad y así poder acceder al Servidor sin problemas.

Una vez establecida la conexión, **el cliente tiene que mandar en un único mensaje su nombre, DNI (como se ha usado para generar el código OTP en el punto anterior)**

y código OTP, separados por el carácter & Se muestran un par de líneas de código (similares a las que deben estar en su código) para aclarar esto:

```
otp = input ("ingresa codigo OTP: ")
Message = "Pepito Grillo&123456&"+otp
```

A continuación, el cliente tiene que esperar un mensaje de confirmación, mostrarlo en pantalla y cerrar la conexión. El mensaje de confirmación le indicará si todo ha sido correcto.

El servidor realizado por los profesores, por su parte, comprobará que el código OTP es válido y almacenará en un fichero esta información, junto con la IP del cliente, para tener un registro de que se ha realizado la práctica. Puede ser útil guardar pruebas de que la práctica se ha realizado correctamente, por ejemplo con una imagen de la pantalla donde se vea que se ha recibido el mensaje de confirmación, por si hubiera algún tipo de problema con la nota de la practica recibida.

Las prácticas son individuales, y cada alumno tiene que emplear su propio dni (sin letras) como password para la generación de códigos OTP. El uso de un dni de otro alumno equivale a haber copiado la práctica y, por tanto, se le aplicará la Normativa de Integridad Académica (NIA) de la ETSIT.