



UNIVERSITAT  
POLITÈCNICA  
DE VALÈNCIA



# **Implementación de una Agente de Usuario de correo electrónico TCP/IP con Python.**

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

## Objetivos

El objetivo de la práctica consiste en comprender cual es la sintaxis de una dirección de correo electrónico, el formato de un mensaje electrónico (RFC 822 y MIME) y el funcionamiento del protocolo de envío de correos electrónicos SMTP utilizando las librerías de las que dispone el lenguaje de programación de alto nivel Python.

Para ello vamos a realizar una serie de tareas en el que se utilizaran las diferentes librerías de las que dispone Python.

### 1 Dirección de correo electrónico

Para que se envíe un correo electrónico es necesario que el remitente especifique quién es el destinatario y proporcione la suficiente información para indicar cómo se puede localizar al destinatario.

Un elemento que diferencia al correo electrónico de otros sistemas es que la comunicación está orientada al usuario, es decir, el correo electrónico se envía de un usuario a otro en lugar de una maquina a otra como se transfiere un archivo mediante FTP. Por lo que no podemos utilizar números de puerto y direcciones IP como la mayoría de los protocolos de aplicación (FTP, SSH, ...) para identificar el destinatario final.

Dado que el correo electrónico se basa en el usuario necesitamos un esquema de direccionamiento distinto que especifique dos piezas de información: quién es el usuario y en qué servidor de correo se ubica su buzón de correo electrónico.

En el sistema de correo electrónico TCP/IP, las direcciones de correo electrónico constan de dos componentes (nombre de usuario y el nombre de dominio) unidos mediante el símbolo @ de la forma: `<usuario>@<nombre de dominio>`. A partir de la dirección de correo electrónico, el agente de usuario será capaz de determinar la ubicación del servidor en el que se ubica el buzón de correo electrónico mediante una petición DNS de tipo MX (*Mail Exchanger*) al servidor DNS del dominio.

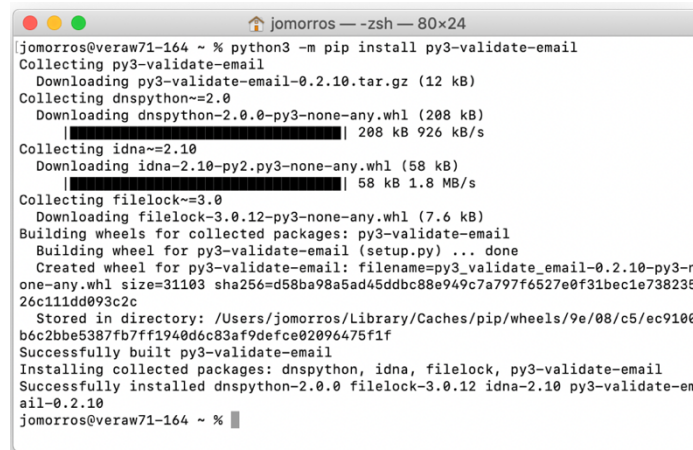
El formato de `<nombre de dominio>` sigue las reglas de sintaxis del sistema DNS, la cual especifica que las etiquetas sólo pueden contener letras y números, y puntos para separar las etiquetas. En cambio, el formato de la parte de `<usuario>` tiene un formato menos restrictivo, permitiendo caracteres especiales como: `!,., #, $, %, &, ', *, +, -, /, =, ?, ^, _', {, |, }, ~`. Además, se permite el uso de otros caracteres ASCII en el `<usuario>` si este se encuentra entrecomillado. Para más información se recomienda revisar las RFC 5322 (secciones 3.2.3 y 3.4.1) y RFC 5321, junto con el documento explicativo que se encuentra en la RFC 3696.

Py3-validate-email es un paquete para Python que permite comprobar si un correo es válido, tiene el formato adecuado y realmente existe<sup>1</sup>. Para poderlo utilizar en nuestras aplicaciones Python es necesario instalarlo, para ello (en función del sistema operativo el que estéis trabajando) ejecuta uno de los siguientes comandos:

---

<sup>1</sup> Para más información de todas sus funciones consultar la página web del proyecto <https://pypi.org/project/py3-validate-email/>

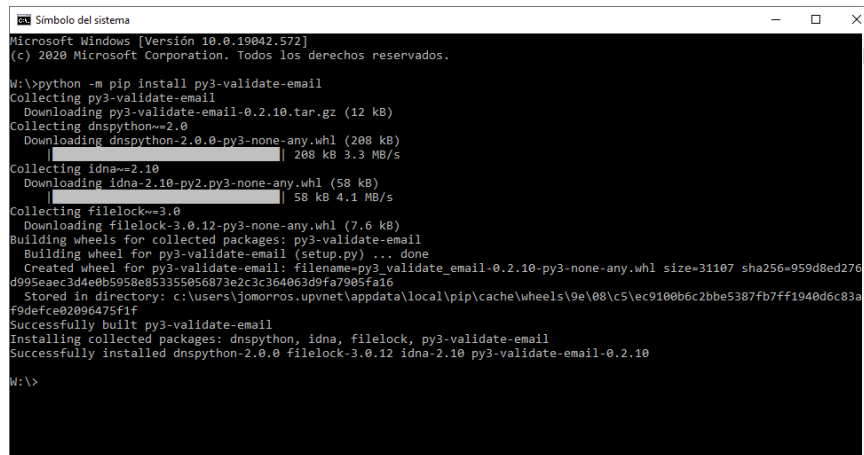
- Linux/macOS (ver Fig. 1): Abra una ventana de Terminal y escriba el comando `python3 -m pip install py3-validate-email`



```
jomorros@veraw71-164 ~ % python3 -m pip install py3-validate-email
Collecting py3-validate-email
  Downloading py3-validate-email-0.2.10.tar.gz (12 kB)
Collecting dnspython~=2.0
  Downloading dnspython-2.0.0-py3-none-any.whl (208 kB)
    | 208 kB 926 kB/s
Collecting idna~=2.10
  Downloading idna-2.10-py2.py3-none-any.whl (58 kB)
    | 58 kB 1.8 MB/s
Collecting filelock~=3.0
  Downloading filelock-3.0.12-py3-none-any.whl (7.6 kB)
Building wheels for collected packages: py3-validate-email
  Building wheel for py3-validate-email (setup.py) ... done
  Created wheel for py3-validate-email: filename=py3_validate_email-0.2.10-py3-n
  one-any.whl size=31103 sha256=d58ba98a5ad45ddbc88e949c7a797f6527e0f31bec1e738235
  26c11dd093c2c
  Stored in directory: /Users/jomorros/Library/Caches/pip/wheels/9e/08/c5/ec9100
  b6c2bbe5387fb7ff1940d6c83af9defce02096475f1f
Successfully built py3-validate-email
Installing collected packages: dnspython, idna, filelock, py3-validate-email
Successfully installed dnspython-2.0.0 filelock-3.0.12 idna-2.10 py3-validate-em
ail-0.2.10
jomorros@veraw71-164 ~ %
```

Fig. 1. Instalación de una librería utilizando pip en Linux/macOS.

- Windows (ver Fig. 2): Abra la línea de comandos ejecutando el programa `cmd.exe` y escriba el comando `py -m pip install py3-validate-email`



```
Microsoft Windows [Versión 10.0.19042.572]
(c) 2020 Microsoft Corporation. Todos los derechos reservados.

C:\>py -m pip install py3-validate-email
Collecting py3-validate-email
  Downloading py3-validate-email-0.2.10.tar.gz (12 kB)
Collecting dnspython~=2.0
  Downloading dnspython-2.0.0-py3-none-any.whl (208 kB)
    | 208 kB 3.3 MB/s
Collecting idna~=2.10
  Downloading idna-2.10-py2.py3-none-any.whl (58 kB)
    | 58 kB 4.1 MB/s
Collecting filelock~=3.0
  Downloading filelock-3.0.12-py3-none-any.whl (7.6 kB)
Building wheels for collected packages: py3-validate-email
  Building wheel for py3-validate-email (setup.py) ... done
  Created wheel for py3-validate-email: filename=py3_validate_email-0.2.10-py3-n
  one-any.whl size=31107 sha256=959d8ed276
  d995eac3d4e0b5958e853355056873e2c3c364063d9fa7905fa16
  Stored in directory: c:\users\jomorros\appdata\local\pip\cache\wheels\9e\08\c5\ec9100b6c2bbe5387fb7ff1940d6c83a
  f9defce02096475f1f
Successfully built py3-validate-email
Installing collected packages: dnspython, idna, filelock, py3-validate-email
Successfully installed dnspython-2.0.0 filelock-3.0.12 idna-2.10 py3-validate-email-0.2.10

C:\>
```

Fig. 2. Instalación de una librería Python utilizando pip en Windows.

## 1.1 Validación de un correo electrónico

Utilizando un editor de código, por ejemplo Visual Studio Code, cree un archivo Python con nombre `valida.py` que contenga el siguiente código:

```
1. from validate_email import validate_email
2.
3. email_from="example@example.com"
4. print("Introduzca un email para comprobar si es correcto")
5. email_to=input("Destinatario:")
6. es_valido=validate_email(email_address=email_to, check_format=True, ch
  eck_blacklist=False,check_dns=False, check_smtp=False)
7. while (es_valido==False):
8.     #La dirección de correo es incorrecta
9.     print("Dirección Incorrecta.Introduzca la dirección de destino de
  nuevo.")
10.    email_to=input("Destinatario:")
11.    es_valido=validate_email(email_address=email_to, check_format=True
  , check_blacklist=False,check_dns=False, check_smtp=False)
```

La explicación del código es bien sencilla. En la línea 1 importamos la función `validate_email` de la librería que acabamos de instalar mientras que en la línea 6 llamamos a esta función, con los siguientes parámetros:

- `email_address`: dirección de correo a verificar.
- `check_format`: Si el valor es `True` comprueba si la dirección de correo electrónico introducido tiene una sintaxis conforme al estándar.
- `check_blacklist`: Si el valor es `True` comprueba si la dirección de correo electrónico esta incluida en alguna lista negra de dominios debido a que son origen del envío de Spam. La lista se descarga de <https://github.com/disposable-email-domains/disposable-email-domains>.
- `check_dns`: Si el valor es `True` comprueba los registros DNS de tipo MX y verifica si el nombre de dominio es válido (dispone de un servidor de correo electrónico entrante) y la dirección existe en dicho dominio. Para que las comprobaciones sean más rápidas es aconsejable dejar el valor en `FALSE` como se muestra en la imagen.
- `check_smtp`: Si el valor es `True` comprueba que la dirección electrónica a verificar existe. Para que las comprobaciones sean más rápidas es aconsejable dejar el valor en `FALSE` como se muestra en la imagen.

Si la dirección no es válida se vuelve a solicitar al usuario que introduzca una dirección de correo válida.

Para comprobar el funcionamiento de esta aplicación, modifique la línea 3 para indicar su correo electrónico de la universidad, ejecute el código y pruebe con algunos correos electrónicos (p.ej. `fulanito@teleco.upv.es`, `menganito@derecho.upv.es`, `..fulanito@teleco.upv.es`, `prueba..prueba@derecho..upv.es`, etc). Reflexione porque en uno casos ha dado error y porque en otros no.

## 2 Envío de un correo electrónico utilizando Python.

El sistema de correo electrónico se basa en formatos de mensajes estándar para garantizar que todos los mensajes tienen la misma forma y estructura. Esto permite que todos los dispositivos del sistema de correo electrónico sean capaces de leer y entender los mensajes, y permitir que el correo electrónico TCP/IP funcione en diferentes tipos de sistemas.

Para facilitar la tarea de enviar correos electrónicos, Python incluye la librería `email` para gestionar mensajes de correo electrónico manteniendo la máxima compatibilidad posible con las RFC que definen tanto el formato de mensaje de correo electrónico IMF (RFC 822 y posteriores actualizaciones), como el formato de mensaje MIME (RFC 2045, RFC 2046, RFC 2047, RFC 2048, RFC 2049, y posteriores actualizaciones).

Sin embargo, la librería `email` no incluye ninguna función para el envío de correo electrónico. Para enviar correos electrónicos utilizando el protocolo de aplicación SMTP (RFC 821 y posteriores actualizaciones) vamos a utilizar la librería `smtplib` la cual implementa un cliente SMTP.

## 2.1 Formato de mensaje IMF

En este apartado vamos a implementar una aplicación en Python que nos permita enviar correos electrónicos utilizando el formato IMF, más conocido por RFC 822 por el estándar que lo define.

Entre las principales características de este formato podemos destacar:

- El mensaje está formado por dos secciones: Cabecera del Mensaje (que indica emisor, receptor e información del mensaje) y Cuerpo de Mensaje (que contiene el texto que el emisor desea transmitir al receptor).
- La cabecera la forman un conjunto de campos, cada uno en una línea, con un formato rígido (<header name>:<header value>). La RFC 822 especifica muchos tipos de campos que pueden incluirse en un mensaje algunos de ellos obligatorios, otros pese a no serlo se incluyen en todos los mensajes y finalmente hay opcionales que aparecen en determinadas circunstancias.
- Los mensajes están formados por líneas de texto codificados en US-ASCII terminados por los caracteres Retorno de Carro y Salto de línea (CR+LF).
- Cada línea de texto tiene una longitud máxima de 998 caracteres, aunque se recomienda un máximo de 78.
- Para separar la Cabecera del Cuerpo del Mensaje se usa una línea vacía.

Un ejemplo del formato RFC 822 es el siguiente:

```
To: profesor_DST@upv.es
From: rector@upv.es
Subject: AVISO
Message-ID: <xxxxx-x-xxx-xxxxx@upv.es>
Date: Thu, 5 Nov 2020 01:21:33 +0100
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64)
```

Se suspenden las clases

A continuación, vamos a crear un archivo en Python que con el nombre 822.py con el siguiente código:

```
1. from validate_email import validate_email
2. from textwrap import TextWrapper
3. # Importamos los módulos que necesitamos de la librería email
4. from email.message import EmailMessage
5.
6. # Importamos la librería smtplib
7. import smtplib
8.
9. #Creamos el objeto de la clase EmailMessage
10. msg=EmailMessage()
11.
12. #Cabecera
13. email_from = "example@@example.com"
14.
15. email_to=input("Destinatario: ")
```

```

16. es_valido=validate_email(email_address=email_to, check_format=True, check_blacklist=False, check_dns=False, check_smtp=False)
17. while (es_valido==False):
18.     #La dirección de correo es incorrecta
19.     print("Dirección Incorrecta. Introduzca la dirección de destino de nuevo.")
20.     email_to=input("Destinatario: ")
21.     es_valido=validate_email(email_address=email_to, check_format=True, check_blacklist=False, check_dns=False, check_smtp=False)
22.
23. msg.add_header('From',email_from)
24. msg.add_header('To',email_to)
25. msg.add_header('Subject',input("Asunto:"))
26.
27. #Cuerpo del mensaje
28. print("Introduzca el contenido del mensaje por ahora solo US-ASCII. Finalice con un \".\" en una línea vacía")
29. contents = []
30. wrapper = TextWrapper(width=78)
31. while True:
32.     line = input()
33.     if (line == '.' ):
34.         break
35.     #Ajustamos las líneas con tamaño máximo 78 caracteres
36.     if len(line)<=78:
37.         contents.append(line)
38.     else:
39.         lineas=wrapper.wrap(text=line)
40.         for i in lineas:
41.             contents.append(i)
42.
43. #Creamos el cuerpo del mensaje
44. body='\n'.join(contents)
45. msg.set_content(body)
46. #Enviamos el correo electrónico utilizando el protocolo SMTP
47. try:
48.     #Establecemos la conexión con el servidor smtp de la UPV
49.     mailServer = smtplib.SMTP("smtp.upv.es", 25)
50.
51.
52.
53.     #Enviamos el mensaje
54.     mailServer.sendmail(email_from,email_to,msg.as_string())
55.     #Cierre de la conexión
56.     mailServer.quit()
57.     print("Correo electrónico enviado")
58. except smtplib.SMTPException as e:
59.     print("Se ha producido un error STMP: "+str(e))
60.     print("Correo electrónico no enviado")

```

La explicación del código es sencilla. En primer lugar, vamos a importar las clases y funciones que vamos a utilizar en nuestro programa (líneas 1 a 7). Además de la función `validate_email` vamos a necesitar:

- `TextWrapper`: Clase que entre otras tareas permite ajustar el tamaño de las líneas de un párrafo a un número determinado de caracteres.

- `EmailMessage`: Clase de la librería `email` incluida en el módulo `email.message`. Esta clase proporciona la funcionalidad principal para crear un mensaje de correo electrónico en el formato RFC 828
- `smtplib`: Clase que define un cliente SMTP en Python.

En la línea 10 creamos un objeto de la clase `EmailMessage`, este objeto contendrá los campos de la cabecera del mensaje y el cuerpo del mensaje que queremos enviar. De hecho, tras solicitar al usuario que introduzca por teclado el destinatario del correo electrónico (utilizando el mismo código fuente visto en el apartado 1.1). Hemos creado la cabecera del mensaje utilizando las siguientes funciones (líneas 23-25):

- `msg.add_header('From', email_from) .`
- `msg.add_header('To', email_to) .`
- `msg.add_header('Subject', input("Asunto:")) .`

Si deseáramos incluir un mayor número de cabeceras las añadiríamos procediendo del mismo modo.

A continuación, deberíamos añadir el cuerpo del mensaje. Para ello, entre las líneas 28 y 41 se ha programado la introducción por teclado de éste. Si nos fijamos en más detalle vamos a utilizar tres elementos:

- Una lista (`contents`).
- Un objeto de la clase `TextWrapper` que nos permitirá ajustar el tamaño de las líneas a 78 caracteres (`wrapper`).
- Una cadena (`line`)

Vamos a leer los datos introducidos por el teclado línea a línea, si el tamaño de la línea es inferior a 78 caracteres la añadimos al final de la lista con el comando `append`, en caso contrario con el comando `wrap` se subdivide la línea en varias cadenas de caracteres con un tamaño máximo de 78 caracteres las cuales se añadirán a nuestra lista `contents`.

Al final, cómo el cuerpo de un mensaje tiene que ser una cadena de caracteres compuesta de líneas acabadas en (CR+LF) con el comando `body='\n'.join(contents)` se concatenan todas las cadenas de texto de la lista siendo el elemento que las concatena los caracteres CR+LF (`'\n'`).

Una vez definido el cuerpo del mensaje este se añade al objeto de la clase `EmailMessage` (línea 44):

- `msg.set_content(body) .`

Una vez definido el mensaje a enviar, entre las líneas 47 a 60 se ha programado el envío del mensaje de correo electrónico utilizando el protocolo SMTP. En este bloque se pueden destacar el uso de las siguientes funciones:

- `mailServer=smtplib.SMTP("host", número_puerto)`. Crea un objeto denominado `mailserver` de la clase `smtplib` que encapsula la conexión SMTP. Tomando como parámetros de entrada el nombre de servidor de correo electrónico de salida y el número de puerto TCP.
- `mailServer.sendmail(email_from, email_to, msg.as_string())`. Envía el correo electrónico toma como parámetros de entrada la dirección

de correo electrónico del remitente (**MAIL FROM:**), la dirección de correo electrónico del destinatario (**RCPT TO:**) y el cuerpo del mensaje como caracteres de texto (**DATA**).

- `mailServer.quit()`. Termina la sesión SMTP y cierra la conexión. Devuelve el resultado del comando SMTP **QUIT**.

Estas instrucciones se han incluido dentro de un bloque de código Python `try-except`. El comando `try` lo podemos traducir por “Intenta ejecutar este código”. Mientras que el comando `except` viene a indicar “Captura cualquier error”. A esta técnica de programación se le denomina **manejo de excepciones**.

El manejo de excepciones permite controlar los errores ocasionados durante la ejecución de un programa informático evitando su finalización anormal como consecuencia de una situación no esperada. De hecho, cuando ocurre cierto tipo de error definido con la sentencia `except`, el programa reacciona ejecutando un fragmento de código que resuelve la situación (por ejemplo: mostrando un mensaje de error o devolviendo un valor por defecto).

Para probar el funcionamiento del programa anterior, modifique la línea 13 para indicar su correo electrónico y pruebe a enviar varios correos electrónicos. Recuerde que si queremos utilizar el servidor de correo electrónico saliente de la UPV es necesario estar conectado a UPVNET, realizando una VPN si estamos fuera del campus, o en caso contrario la aplicación mostrará un error.

**Importante: Use su dirección de alumno de la UPV.**

#### 2.1.1 Seguridad

La librería `smtpplib` también tiene implementada la extensión **STARTTLS** (RFC3207), esta extensión permite que una sesión sea cifrada por medio de TLS/SSL. En este caso, será necesario establecer la sesión con las extensiones de SMTP. Por lo que habrá que añadir las siguientes líneas entre el establecimiento de la conexión del servidor y el envío del mensaje (en las líneas 50 y 51):

- `mailServer.ehlo()`. Para indicar que se admiten las extensiones SMTP. Equivalente al comando de SMTP **EHLO**.
- `mailServer.starttls()`. Pone la conexión SMTP en modo TLS (Seguridad de la capa de transporte). Equivale al comando de SMTP **STARTTLS**.

A continuación, deberá modificar el programa 822.py para enviar correos electrónicos cifrados y enviar varios correos electrónicos para probar el funcionamiento de este programa. Recuerde que si queremos utilizar el servidor de correo electrónico saliente de la UPV es necesario estar conectado a UPVNET en caso contrario la aplicación mostrará un error.

**Importante: Use su dirección de alumno de la UPV.**

#### 2.1.2 Autenticación

Los administradores de los servidores de correo saliente deben imponer cierto control sobre qué clientes pueden utilizar el servidor. Esto les permite lidiar con abusos (por ejemplo: el spam). Principalmente se han utilizado dos soluciones:



- Restringir el acceso por ubicación. Bajo este sistema el servidor de correo saliente de una empresa no permitirá el acceso de usuarios que estén fuera de su red (método que utiliza la UPV).
- Autenticación del cliente (**SMTP AUTH**). En lugar de restringir por ubicación, los servidores SMTP que implementan las extensiones SMTP, permiten la autenticación de los clientes mediante un mecanismo de control de acceso.

La librería `smtpplib` permite enviar correos electrónicos utilizando el comando **SMTP AUTH**, lo que nos permitiría enviar correos electrónicos desde la cuenta de la UPV (sin la obligación de estar conectados a UPVNET) o desde la cuenta de otro proveedor. Para que un agente de correo pueda utilizar la autenticación SMTP es recomendable utilizar el puerto TCP 587 (RFC 2476 y posteriores actualizaciones). Este puerto está reservado para realizar conexiones SMTP utilizando de forma obligatoria una sesión SSL/TLS.

Por lo ello una vez establecida la sesión cifrada, será necesario autenticarnos con el servidor de correo para ello utilizaremos la siguiente función de la librería `smtpplib`:

- `mailServer.login(user="usuario",password="xxxx")` . Inicia la sesión en un servidor de correo saliente que requiera autenticación. Los argumentos son el nombre de usuario, normalmente la dirección de correo electrónico del usuario, y la contraseña para autenticarse.

Para probar esta extensión, modifique el programa desarrollado en el apartado 2.1.1 para utilizar el puerto 587 en lugar del puerto 25, e incorpore el comando de autenticación después de abrir la sesión TLS (en la línea 52. En el campo `user`, deberá poner su dirección de correo electrónico y en el campo `password` deberá indicar su contraseña de correo electrónico NO EL PIN. Dado que puede que no la recordéis, porque accedéis a través de la intranet o tenéis redireccionado el correo electrónico a otras cuentas (p.ej. Gmail.com), en el siguiente enlace se indican las instrucciones que debéis seguir para cambiarla: <https://www.upv.es/contenidos/INFOACCESO/infoweb/infoacceso/dat/839429normalc.html>.

Que se resume en: Entre en su Intranet con su DNI y PIN (número de 4 dígitos) y en el apartado 'Servicios'-> 'Correo Electrónico'-> 'Utilidades'-> 'Cambiar contraseña de correo' puede cambiarla.

Si está en casa pruebe realizar esto desactivando la VPN.

## 2.2 Formato MIME (Multipurpose Internet Mail Extensions)

El uso de caracteres en formato ASCII facilitó la creación, procesado y lectura de mensajes, lo que contribuyó a su éxito. Sin embargo, mientras que esta codificación es perfecta para enviar correos con texto, no proporciona la flexibilidad necesaria para soportar otro tipo de datos. Con el fin de los que los emails pudieran transportar algo más que texto en formato ASCII estándar se creó el formato MIME. Es decir, utilizando MIME, es posible adjuntar archivos de cualquier tipo y texto que use caracteres no estándar cumpliendo el formato de mensaje RFC822.

La librería `email` que hemos utilizado en el apartado 2.1 permite construir mensajes de correo electrónico y ocultando los detalles de las diversas RFC que rigen la aplicación. Conceptualmente la librería debería poder tratar el mensaje de correo electrónico como un árbol estructurado de texto Unicode y archivos adjuntos binarios, sin tener que preocuparse por cómo se representan estos cuando se convierten en líneas de texto. Sin embargo, en la práctica, es necesario conocer al menos algunas de las reglas que rigen los mensajes MIME y su estructura, específicamente los nombres y la naturaleza de los “tipos de medios” MIME y cómo se identifican dentro de los mensajes con estructura compleja (mensajes multiparte).

Por ello en este apartado nos vamos a centrar en el envío de un mensaje multiparte en formato MIME utilizando las librerías que hemos utilizado en los apartados anteriores. De hecho, vamos a desarrollar un programa en Python que nos permita enviar un correo electrónico con una imagen adjunta. Para no desaprovechar el trabajo realizado hasta el momento vamos partir del programa implementado en el apartado 2.1.2 guardándolo como `MIME.py`.

El primer cambio que debemos realizar es importar las siguientes clases de la librería `email`:

- `MIMEMultipart`: Que nos permite definir instancias de mensajes MIME de tipo multiparte.
- `MIMEText`: Que nos permite definir instancias de parte de mensajes MIME cuyo Content-Type es de tipo texto.
- `MIMEImage`: Que nos permite definir instancias de parte de mensajes MIME cuyo Content-Type es de tipo Imagen.
- `MIMEApplication`: Que permite definir instancias de parte de mensajes cuyo Content-Type es de tipo Aplicación.

Otras librerías que también deberemos importar son:

- `sys`: Librería que permite acceder a algunas variables y funciones que interactúan con el intérprete de comandos.
- `encode_base64`: Librería que incluye el codificador base64 utilizado por los constructores de clases `MIMEAudio`, `MIMEImage` (entre otros) para proporcionar la codificación utilizada por estos tipos de medios.

A continuación, vamos a declarar el objeto mensaje. En este caso, en lugar de declarar un objeto de la clase `EmailMessage()` se declarará de la clase `MIMEMultipart("mixed")`. Otras posibles opciones definidas en la RFC 2046 y que no vamos a estudiar en esta práctica son: `"alternative"`, `"parallel"`, etc. De forma opcional, a la hora de declarar el objeto se puede definir el delimitador o frontera entre partes. Si no se define, este delimitador se calculará cuando se vaya a enviar el correo electrónico.

- `msg=MIMEMultipart("mixed", "frontera")`

Otro cambio, que se habrá de realizar es a la hora de crear el cuerpo del mensaje, al ser un mensaje en formato MIME, vamos a tener varias partes en las que cada una de las partes del cuerpo se procesa y se le añaden sus propias cabeceras, como Content-Type, Content-ID y Content-Transfer-Encoding. Es decir, vamos a tener que crear cada

subparte de forma independiente. Para la subparte que es texto será necesario utilizar los siguientes comandos:

- `parte_texto=MIMEText(text, subtipo, charset)`. Constructor de la clase `MIMEText` para construir subpartes MIME de tipo Texto. Tanto, el subtipo (i.e. `plain`) como la codificación de los caracteres (i.e.: `utf-8`) se pasa como parámetro.
- `msg.attach(parte_texto)`. Método de la clase `MIMEMultipart` para añadir subpartes al mensaje.

Con respecto a la imagen, el proceso es un poco más complejo, puesto que habrá que abrir el archivo, construir el objeto subparte, definir las cabeceras de la parte y cerrar el archivo. Para ello será necesario utilizar los siguientes comandos:

```
#Adjuntamos la imagen
fName="filename.jpg"
subtipo="jpg"
try:
    file=open(fName, 'rb')
    adjunto = MIMEImage(file.read(), subtipo)
    adjunto.add_header('Content-Disposition', 'attachment; filename = \''+fName+'\'')
    msg.attach(adjunto)
    file.close()
except OSError as e:
    print(e)
    print("Correo no enviado")
    sys.exit()
except:
    print("Correo no enviado")
    sys.exit()
```

En el bloque de código anterior, vamos a enviar como adjunto la imagen "filename.jpg" definido en la variable `fName` que se encuentra en la ruta dónde estamos ejecutando el programa en Python siendo su subtipo "jpg". A continuación lo abriremos en modo lectura y binario `'rb'` y crearemos un objeto de la clase `MIMEImage` con el subtipo almacenado en la variable `subtipo` y el contenido del archivo. Este objeto contendrá la segunda parte de nuestro mensaje MIME. A continuación, deberemos definir cuáles son las cabeceras de esta subparte, en este caso vamos a utilizar la cabecera adicional `Content-Disposition` e indicaremos cual es el nombre del adjunto. Finalmente, añadiremos esta subparte al mensaje MIME. Si se produce cualquier tipo de error, por ejemplo: no existe el fichero, el programa nos mostrará un mensaje de error y el mensaje no se enviará. El código completo se muestra a continuación.

```
1. from validate_email import validate_email
2. from textwrap import TextWrapper
3. import sys
4. # Importamos los módulos que necesitamos de la librería email
5. from email.mime.multipart import MIMEMultipart
6. from email.mime.text import MIMEText
7. from email.mime.image import MIMEImage
8. from email.mime.application import MIMEApplication
9. from email.encoders import encode_base64
10.
```

```

11. # Importamos la librería smtplib
12. import smtplib
13.
14. # Creamos el objeto de la clase mensaje MIME Multipart
15. msg=MIMEMultipart("mixed","frontera")
16.
17. #Cabecera
18. email_from = "example@example.com"
19.
20. email_to=input("Destinatario: ")
21. es_valido=validate_email(email_address=email_to, check_format=True, check_blacklist=False, check_dns=False, check_smtp=False)
22. while (es_valido==False):
23.     #La dirección de correo es incorrecta
24.     print("Dirección Incorrecta. Introduzca la dirección de destino de nuevo.")
25.     email_to=input("Destinatario: ")
26.     es_valido=validate_email(email_address=email_to, check_format=True, check_blacklist=False, check_dns=False, check_smtp=False)
27. msg.add_header('From',email_from)
28. msg.add_header('To',email_to)
29. msg.add_header('Subject',input("Asunto:"))
30.
31. #Parte de texto del mensaje
32. print("Introduzca el mensaje. Puede usar cualquier caracter. Finalice con un \"\\n\" en una línea vacía")
33. contents = []
34. wrapper = TextWrapper(width=78)
35. while True:
36.     line = input()
37.     if (line == '.' ):
38.         break
39.     #Ajustamos las líneas con tamaño máximo 78 caracteres
40.     if len(line)<=78:
41.         contents.append(line)
42.     else:
43.         lineas=wrapper.wrap(text=line)
44.         for i in lineas:
45.             contents.append(i)
46.
47. # Adjuntamos el texto
48. parte_texto=MIMEText('\n'.join(contents),"plain","utf-8")
49. msg.attach(parte_texto)
50.
51. #Adjuntamos la imagen
52. fName="filename.jpg"
53. subtipo="jpg"
54. try:
55.     file=open(fName, 'rb')
56.     adjunto = MIMEImage(file.read(),subtipo)
57.     adjunto.add_header('Content-Disposition',
58.         'attachment; filename = \''+fName+'\'')
59.     msg.attach(adjunto)
60.     file.close()
61. except OSError as e:
62.     print(e)
63.     print("Correo no enviado")
64.     sys.exit()
65. except:

```

```

65.     print("Correo no enviado")
66.     sys.exit()
67.
68. #Enviamos el correo electrónico utilizando el protocolo SMTP
69. try:
70.     #Establecemos la conexión con el servidor smtp de la UPV
71.     mailServer = smtplib.SMTP("smtp.upv.es", 587)
72.     mailServer.ehlo()
73.     mailServer.starttls()
74.     mailServer.login(user=email_from,password="*****")
75.     #Enviamos el mensaje
76.     mailServer.sendmail(email_from,email_to,msg.as_string())
77.     #Cierre de la conexión
78.     mailServer.quit()
79.     print("Correo electrónico enviado")
80. except smtplib.SMTPException as e:
81.     print("Se ha producido un error SMTP: "+str(e))
82.     print("Correo electrónico no enviado")

```

A continuación, para comprobar la comprensión de esta práctica deberá enviar un correo electrónico en formato MIME a su dirección de correo electrónico. Este mensaje estará compuesto de dos partes:

- Un texto en el que se incluya su nombre y DNI.

- Una imagen adjunta que debe estar situado en la misma ruta que el programa en Python. En caso contrario se generará un error.

Una vez haya conseguido enviar un correo electrónico con una imagen como adjunto deberá crear un programa capaz de enviar un correo electrónico que está compuesto de dos partes:

- Un texto en el que se incluya su nombre y DNI.

- Un archivo en formato pdf que también incluya su nombre y DNI.

Para ser capaz de realizarlo, deberá utilizar la clase `MIMEApplication` para construir el objeto con el adjunto en lugar la clase `MIMEImage` e indicar el subtipo adecuado en la variable correspondiente.

Una vez haya comprobado que su programa funciona y manda **el texto y el archivo pdf**, envíelo a la siguiente dirección [practica.dst@gmail.com](mailto:practica.dst@gmail.com) Este correo será la prueba de que se ha realizado la práctica correctamente.