Sending the Email Through an SMTP Server

As we called out, to send emails, our computers use the ***Simple Mail Transfer Protocol (SMTP)***. This protocol specifies how computers can deliver email to each other. There are certain steps that need to be followed to do this correctly. But, as usual, we won't do this manually; we'll send the message using the built-in smtplib Python module. Let's start by importing the module.

```
>>> import smtplib
```

With smtplib, we'll create an object that will represent our mail server, and handle sending messages to that server. If you're using a Linux computer, you might already have a configured SMTP server like postfix or sendmail. But maybe not. Let's create a smtplib.SMTP object and try to connect to the local machine.

```
>>> mail_server = smtplib.SMTP('localhost')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  (...We deleted a bunch of lines here...)
ConnectionRefusedError: [Errno 61] Connection refused
```

Oops! This error means that there's no local SMTP server configured. But don't panic! You can still connect to the SMTP server for your personal email address. Most personal email services have instructions for sending email through SMTP; just search for the name of your email service and "SMTP connection settings".

When setting this up, there are a couple of things that you'll probably need to do: Use a secure transport layer and authenticate to the service using a username and password. Let's see what this means in practice.

You can connect to a remote SMTP server using ***Transport Layer Security (TLS)***. An earlier version of the TLS protocol was called ***Secure Sockets Layer (SSL)***, and you'll sometimes see TLS and SSL used interchangeably. This SSL/TLS is the same protocol that's used to add a secure transmission layer to HTTP, making it HTTPS. Within the smtplib, there are two classes for making connections to an SMTP server: The ***SMTP class*** will make a direct SMTP connection, and the ***SMTP_SSL class*** will make a SMTP connection over SSL/TLS. Like this:

```
>>> mail_server = smtplib.SMTP_SSL('smtp.example.com')
```

If you want to see the SMTP messages that are being sent back and forth by the smtplib module behind the scenes, you can set the debug level on the SMTP or SMTP_SSL object. The examples in this lesson won't show the debug output, but you might find it interesting!

```
mail_server.set_debuglevel(1)
```

Now that we've made a connection to the SMTP server, the next thing we need to do is authenticate to the SMTP server. Typically, email providers wants us to provide a username and password to connect. Let's put the password into a variable so it's not visible on the screen.

```
>>> import getpass
>>> mail_pass = getpass.getpass('Password? ')
Password?
>>>

```

The example above uses the getpass module so that passers-by won't see the password on the screen. Watch out, though; the **mail_pass** variable is still just an ordinary string!

```
>>> print(mail_pass)
It'sASecr3t!
```

Now that we have the email user and password configured, we can authenticate to the email server using the SMTP object's login method.

```
>>> mail_server.login(sender, mail_pass)
(235, b'2.7.0 Accepted')
```

If the login attempt succeeds, the login method will return a tuple of the SMTP status code and a message explaining the reason for the status. If the login attempt fails, the module will raise a SMTPAuthenticationError exception.

If you wrote a script to send an email message, how would you handle this exception?

Sending your message

Alright! We're connected and authenticated to the SMTP server. Now, how do we send the message?

```
>>> mail_server.send_message(message)
{}

```

Okay, well that last bit was pretty easy! We did the hard part first! The send_message method returns a dictionary of any recipients that weren't able to receive the message. Our message was delivered successfully, so send_message returned an empty dictionary. Finally, now that the email is sent, let's close the connection to the mail server.

```
>>> mail_server.quit()
```

And there you have it! We covered a lot in this lesson, so let's recap! First, we constructed an email message by using the built-in email module's EmailMessage class. Next, we added an attachment to our message with the help of the built-in mimetypes module. Finally, we connected to a SMTP server and sent the email using the smtplib module's 's SMTP_SSL class.

Did you have any idea all of this was happening behind a simple email message?

✓ **Completed** **Go to next item**

Like **Dislike** **Report an issue**