Sending email in python

We'll explore the basics of sending email in python

WE'LL COVER THE FOLLOWING

Python provides a couple of really nice modules that we can use to craft emails with. They are the **email** and **smtplib** modules. Instead of going over various methods in these two modules, we'll spend some time learning how to actually use these modules. Specifically, we'll be covering the following:

- The basics of emailing
- How to send to multiple addresses at once
- How to send email using the TO, CC and BCC lines
- How to add an attachment and a body using the email module

|

Let's get started!

Email Basics - How to Send an Email with smtplib

The **smtplib** module is very intuitive to use. Let's write a quick example that shows how to send an email. Save the following code to a file on your hard drive:

```
import smtplib

HOST = "mySMTP.server.com"
SUBJECT = "Test email from Python"
TO = "mike@someAddress.org"
FROM = "python@mydomain.com"
```

```
text = "Python 3.4 rules them all!"

BODY = "\r\n".join((
    "From: %s" % FROM,
    "To: %s" % TO,
    "Subject: %s" % SUBJECT ,
    "",
    text
    ))

server = smtplib.SMTP(HOST)
server.sendmail(FROM, [TO], BODY)
server.quit()
```

We imported two modules, **smtplib** and the **string** module. Two thirds of this code is used for setting up the email. Most of the variables are pretty self-explanatory, so we'll focus on the odd one only, which is **BODY**. Here we use the **string** module to combine all the previous variables into a single string where each lines ends with a carriage return ("/r") plus new line ("/n"). If you print BODY out, it would look like this:

```
'From: python@mydomain.com\r\nTo: mike@mydomain.com\r\nSubject: Test email from Python\r\n\r\
```

After that, we set up a server connection to our host and then we call the smtplib module's **sendmail** method to send the email. Then we disconnect from the server. You will note that this code doesn't have a username or password in it. If your server requires authentication, then you'll need to add the following code:

```
server.login(username, password)
```

This should be added right after you create the server object. Normally, you would want to put this code into a function and call it with some of these parameters. You might even want to put some of this information into a config file. Let's put this code into a function.

```
import smtplib

def send_email(host, subject, to_addr, from_addr, body_text):
    """
    Send an email
    """
    BODY = "\r\n".join((
            "From: %s" % from_addr,
            "To: %s" % to_addr,
```

```
            "Subject: %s" % subject ,
            "",
            body_text

            ))
    server = smtplib.SMTP(host)
    server.sendmail(from_addr, [to_addr], BODY)
    server.quit()

if __name__ == "__main__":
    host = "mySMTP.server.com"
    subject = "Test email from Python"
    to_addr = "mike@someAddress.org"
    from_addr = "python@mydomain.com"
    body_text = "Python rules them all!"
    send_email(host, subject, to_addr, from_addr, body_text)
```

Now you can see how small the actual code is by just looking at the function itself. That's 13 lines! And we could make it shorter if we didn't put every item in the BODY on its own line, but it wouldn't be as readable. Now we'll add a config file to hold the server information and the **from** address. Why? Well in the work I do, we might use different email servers to send email or if the email server gets upgraded and the name changes, then we only need to change the config file rather than the code. The same thing could apply to the **from** address if our company was bought and merged into another.

Let's take a look at the config file (save it as **email.ini**):

```
[smtp]
server = some.server.com
from_addr = python@mydomain.com
```

That is a very simple config file. In it we have a section labelled **smtp** in which we have two items: server and from_addr. We'll use configObj to read this file and turn it into a Python dictionary. Here's the updated version of the code (save it as **smtp_config.py**):

```
import os
import smtplib
import sys

from configparser import ConfigParser

def send_email(subject, to_addr, body_text):
    """
    Send an email
    """
    base_path = os.path.dirname(os.path.abspath(__file__))
    config_path = os.path.join(base_path, "email.ini")
```

```
    if os.path.exists(config_path):
        cfg = ConfigParser()
        cfg.read(config_path)
    else:
        print("Config not found! Exiting!")
        sys.exit(1)

    host = cfg.get("smtp", "server")
    from_addr = cfg.get("smtp", "from_addr")

    BODY = "\r\n".join((
        "From: %s" % from_addr,
        "To: %s" % to_addr,
        "Subject: %s" % subject ,
        "",
        body_text
    ))
    server = smtplib.SMTP(host)
    server.sendmail(from_addr, [to_addr], BODY)
    server.quit()

if __name__ == "__main__":
    subject = "Test email from Python"
    to_addr = "mike@someAddress.org"
    body_text = "Python rules them all!"
    send_email(subject, to_addr, body_text)
```

We've added a little check to this code. We want to first grab the path that the script itself is in, which is what base_path represents. Next we combine that path with the file name to get a fully qualified path to the config file. We then check for the existence of that file. If it's there, we create a **ConfigParser** and if it's not, we print a message and exit the script. We should add an exception handler around the **ConfigParser.read()** call just to be on the safe side though as the file could exist, but be corrupt or we might not have permission to open it and that will throw an exception. That will be a little project that you can attempt on your own. Anyway, let's say that everything goes well and the ConfigParser object is created successfully. Now we can extract the host and from_addr information using the usual ConfigParser syntax.

Now we're ready to learn how to send multiple emails at the same time!