Add an attachment / body using the email module

How to add attachments to your emails in python

WE'LL COVER THE FOLLOWING

- Wrapping Up

Now we'll take what we learned from the previous section and mix it together with the Python email module so that we can send attachments. The email module makes adding attachments extremely easy. Here's the code:

```
import os
import smtplib
import sys

from configparser import ConfigParser
from email import encoders
from email.mime.text import MIMEText
from email.mime.base import MIMEBase
from email.mime.multipart import MIMEMultipart
from email.utils import formatdate

#----------------------------------------------------------------------
def send_email_with_attachment(subject, body_text, to_emails,
                               cc_emails, bcc_emails, file_to_attach):
    """
    Send an email with an attachment
    """
    base_path = os.path.dirname(os.path.abspath(__file__))
    config_path = os.path.join(base_path, "email.ini")
    header = 'Content-Disposition', 'attachment; filename="%s"' % file_to_attach

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

    # extract server and from_addr from config
    host = cfg.get("smtp", "server")
    from_addr = cfg.get("smtp", "from_addr")

    # create the message
```

```
    msg = MIMEMultipart()
    msg["From"] = from_addr
    msg["Subject"] = subject
    msg["Date"] = formatdate(localtime=True)
    if body_text:
        msg.attach( MIMEText(body_text) )

    msg["To"] = ', '.join(to_emails)
    msg["cc"] = ', '.join(cc_emails)

    attachment = MIMEBase('application', "octet-stream")
    try:
        with open(file_to_attach, "rb") as fh:
            data = fh.read()
        attachment.set_payload( data )
        encoders.encode_base64(attachment)
        attachment.add_header(*header)
        msg.attach(attachment)
    except IOError:
        msg = "Error opening attachment file %s" % file_to_attach
        print(msg)
        sys.exit(1)

    emails = to_emails + cc_emails

    server = smtplib.SMTP(host)
    server.sendmail(from_addr, emails, msg.as_string())
    server.quit()

if __name__ == "__main__":
    emails = ["mike@someAddress.org", "nedry@jp.net"]
    cc_emails = ["someone@gmail.com"]
    bcc_emails = ["anonymous@circe.org"]

    subject = "Test email with attachment from Python"
    body_text = "This email contains an attachment!"
    path = "/path/to/some/file"
    send_email_with_attachment(subject, body_text, emails,
                               cc_emails, bcc_emails, path)
```

Here we have renamed our function and added a new argument, **file_to_attach**. We also need to add a header and create a **MIMEMultipart** object. The header could be created any time before we add the attachment. We add elements to the **MIMEMultipart** object (msg) like we would keys to a dictionary. You'll note that we have to use the email module's **formatdate** method to insert the properly formatted date. To add the body of the message, we need to create an instance of **MIMEText**. If you're paying attention, you'll see that we didn't add the BCC information, but you could easily do so by following the conventions in the code above. Next we add the attachment. We wrap it in an exception handler and use the **with** statement to extract the file and place it in our **MIMEBase** object. Finally we add it to the **msg** variable and we send it out. Notice that we have to convert the **msg** to a string in the

sendmail method.

Wrapping Up

Now you know how to send out emails with Python. For those of you that like mini projects, you should go back and add additional error handling around the **server.sendmail** portion of the code in case something odd happens during the process, such as an **SMTPAuthenticationError** or **SMTPConnectError**. We could also beef up the error handling during the attachment of the file to catch other errors. Finally, we may want to take those various lists of emails and create one normalized list that has removed duplicates. This is especially important if we are reading a list of email addresses from a file.

Also note that our from address is fake. We can spoof emails using Python and other programming languages, but that is very bad etiquette and possibly illegal depending on where you live. You have been warned! Use your knowledge wisely and enjoy Python for fun and profit!