SECTION 18: EMAILS WITH PYTHON - 28 minutes, 3 parts
2/3 Sending Emails with Python

- **sending emails with Python**
  - -> this requires multiple steps
    - -> connecting to a server
    - -> confirming connection
    - -> setting a protocol
    - -> logging on
    - -> sending a message

Sending Emails

- **the built-in smtplib library**
  - -> this can be used to make function calls
  - -> to call the function from the smtp lib library in the correct order
  - -> each email provider has a male transfer protocol server
  - -> a domain we connect to when we try and access an email programatically
  - -> gmail is the most popular one

PIERIAN DATA

Complete Python Bootcamp

- To send emails with Python, we need to manually go through the steps of connecting to an email server, confirming connection, setting a protocol, logging on, and sending the message.

- **to connect to the gmail domain name**
  - -> we want to generate an app password, rather than a normal one
  - -> this is a specialised password which tells it there is an app password elsewhere
  - -> in case this application is compromised
  - -> creating a special app password for the Python script
  - -> he imports smtplib
  - **-> then creates an smtp object for a server**
    - -> he defines a variable and uses the SMPT method to create this
    - -> he provides an email and a port number as the arguments for this
    - -> we can pass in any number for the port number

PIERIAN DATA

Complete Python Bootcamp

- Fortunately the built-in smtplib library in Python makes these steps simple function calls.

PIERIAN DATA

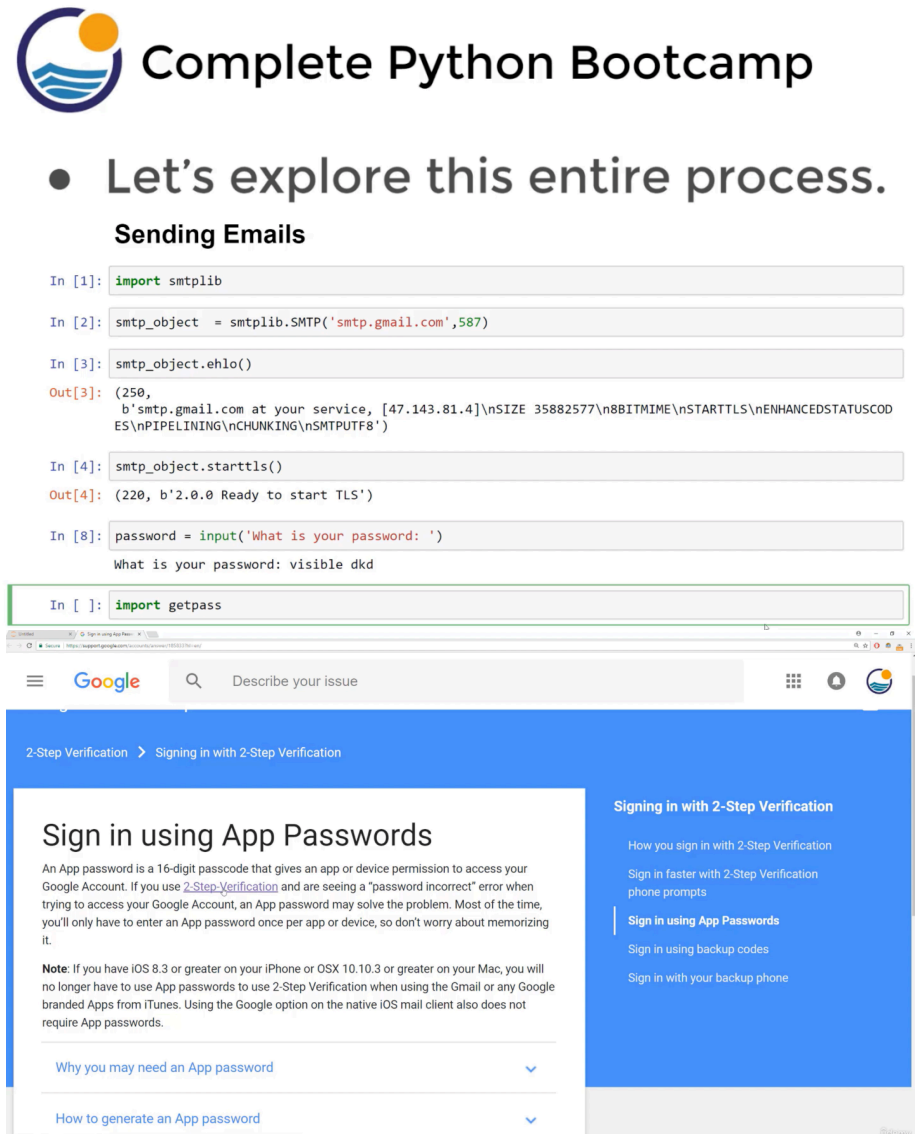
Complete Python Bootcamp

- Each major email provider has their own SMTP (Simple Mail Transfer Protocol) Server.

Provider	SMTP server domain name
Gmail (will need App Password)	smtp.gmail.com
Yahoo Mail	smtp.mail.yahoo.com
Outlook.com/Hotmail.com	smtp-mail.outlook.com
AT&T	smpt.mail.att.net (Use port 465)
Verizon	smtp.verizon.net (Use port 465)
Comcast	smtp.comcast.net

PIERIAN DATA

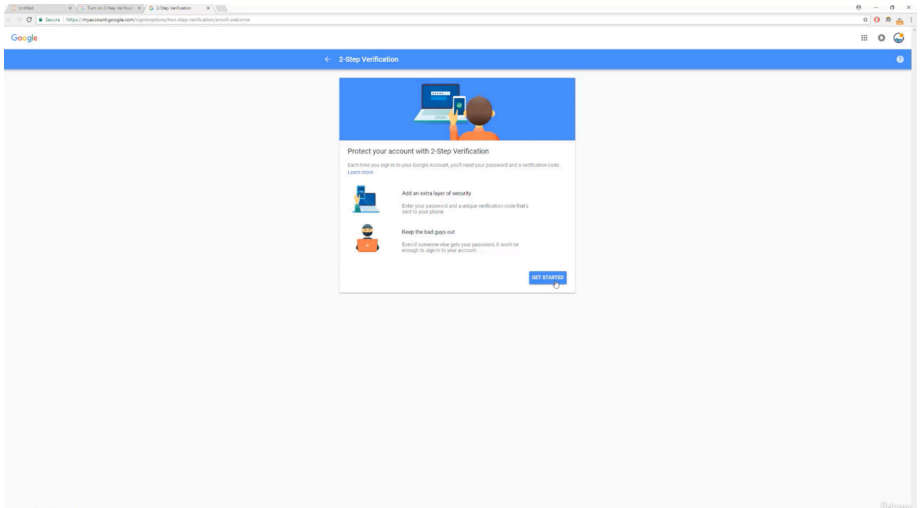
- **-> running a ehlo method**
  - -> this establishes a connection to the server
  - -> this method should be done after creating the object
    - -> otherwise errors can be created
  - -> this returns a code (250 here) which indicates a successful connection
  - -> to initiate the encryption, we grab the object again
  - -> using the .starttls method

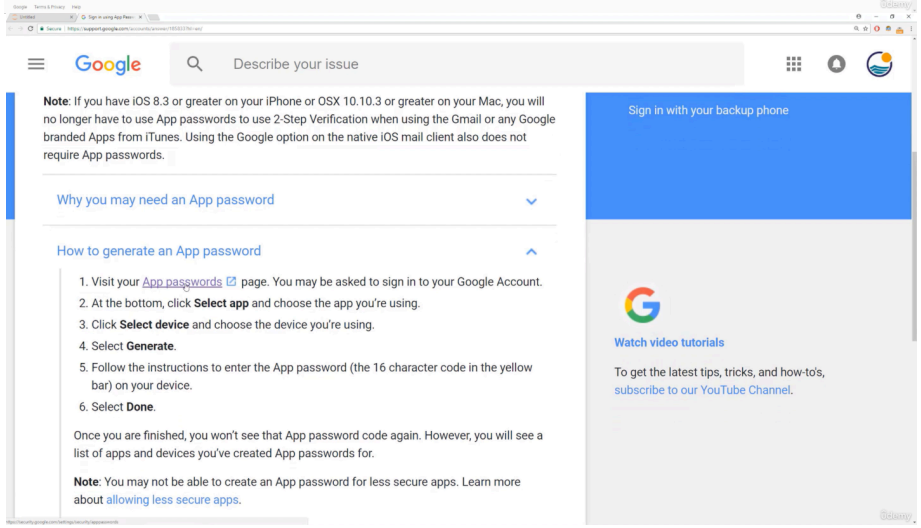


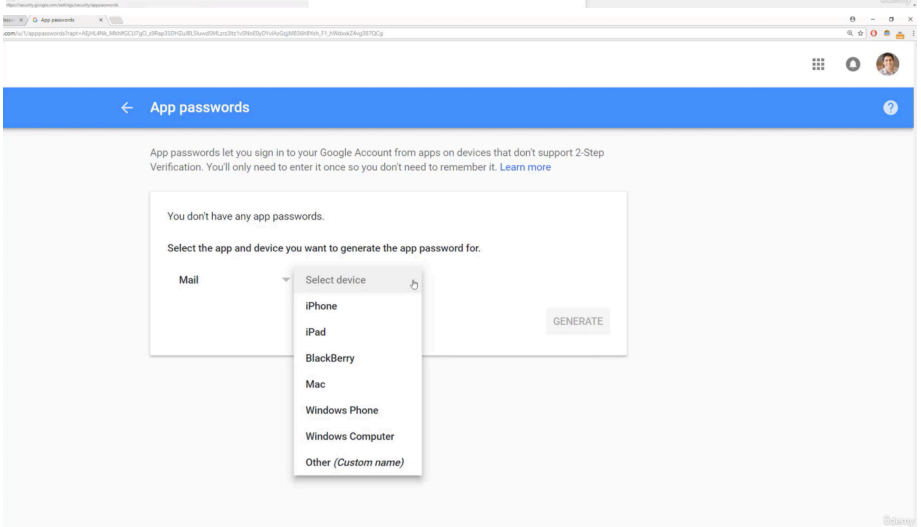
- **-> then setting up the email and passwords**
  - -> don't save the raw string under password in the script
  - -> use the input function instead
  - -> we don't want the password to be in the ipynb without using the getpass method to hide the text

  - **-> example**
    - -> using the input() method in a cell to ask the user for a password
    - -> we can also import the getpass method
    - -> he defines a variable which does this with the getpass module rather than the input method
    - -> this hides the password on the screen by formatting what the user enters into bullet points as they type

- **-> two factor authentication**
  - -> this is for gmail users
  - -> we are generating a user password
  - -> this also requires authentication (two factor in some cases)
  - -> there is a link
  - -> we are connecting to gmail in an ipynb
  - -> a two step verification link
  - -> once we have signed in, we can set the phone number as the second method of verification
  - **-> to send gmail emails in Python, two step verification needs to be turned on for the gmail account we are sending the emails from**

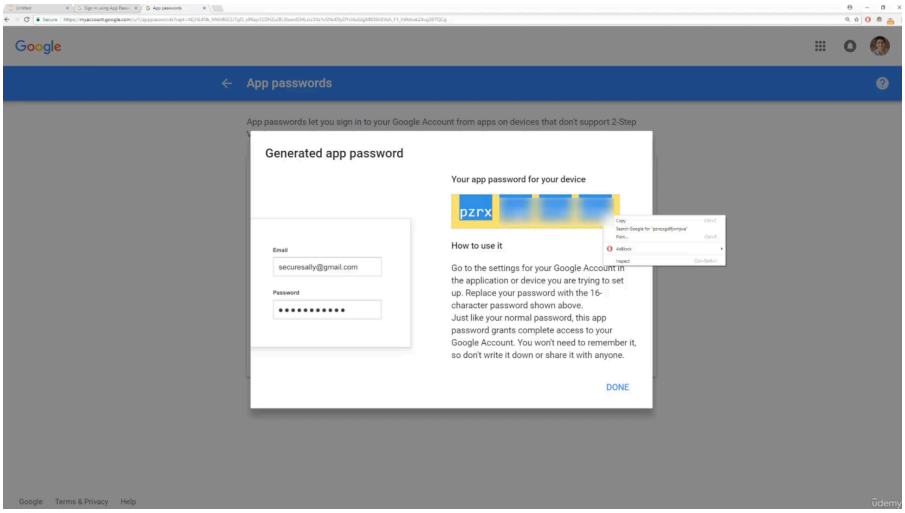






- **-> then setting up the app passwords**
  - -> generating app passwords
  - -> if we have an error message, then two factor authentication isn't turned on
  - -> we have a two app page
  - -> we want to use the mail app, and then select the app password for it
  - -> this generates an app password
  - -> this is the password which we are going to use for the gmail login from the ipynb file

- **-> setting up the email**
  - -> he defines a new variable using the .getpass() method
  - -> he does this twice (once for email and another for password)
  - -> he then uses the .login() method with both of these variables (email and password) as its arguments
  - -> running this asks the user to enter an email
  - -> since this was defined using the .getpass() method, the text which the user enters for this is bullet pointed out (as it would be if you typed a password on the screen)

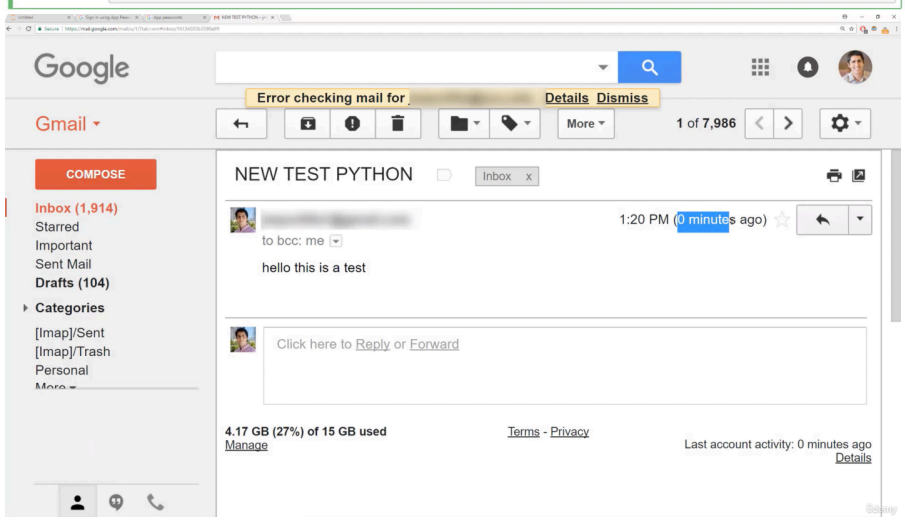



```
In [10]: password = getpass.getpass('Password please: ')
         Password please: ·····

In [11]: email = getpass.getpass("Email: ")
         password = getpass.getpass("Password: ")
         smtp_object.login(email,password)
         Email: ·····
         Password: ·····
Out[11]: (235, b'2.7.0 Accepted')

In [ ]:  from_address = email
         to_address = email
         subject = input("enter the subject line: ")
         message = input("enter the body message: ")
         msg = "Subject: "+subject+'\n'+message

         smtp_object.sendmail(from_address,to_address,msg)
```



  - -> running this cell then asks the user to enter a password after the email
  - -> since the email and password variables in this case were defined using the .getpass() method, this is equivalent to using the input() method - except with the text which the user enters being hidden on the screen as if it would be if you were to type a password on the screen
  - -> so, the .getpass() method is used to ask for an input while covering up what the user enters on the screen
  - -> and then the .login() method is used to ask the user to enter an input for this
  - -> in this case, we are asking for an input for both email and password
  - -> for the second variable whose value is asked for (password), the password that he pastes into the ipynb file for this is the app password which he generated from the previous stage in this
  - -> the code that this returns ('Accepted') indicates that this was successfully done
  - -> this process has to be repeated if you leave the ipynb file and then come back to it
  - -> each time this is done, a new app password must be generated to do so

- **-> to send an email in Python after this is done**
  - -> he defines two new variables (one email to send the message from and one to)
  - -> then he defines a new variable called `subject`, which asks for an input for the subject line of the of the email which we want to send
  - -> he repeats this process, but for the message of the email
  - -> in the code cell in the ipynb file, he is typing all of the information that the email would contain
  - -> he then defines another variable which combines all of these other variables into the content for the message
  - -> he then uses the .sendmail() method with three arguments (the address we want to send the email from, the address we want to send the email to and the message that we want to send)
  - -> then runs the cell
  - -> this first asks the user to enter the subject of the email
  - -> then the message that the email should send in its body
  - -> running this sends the email
  - -> we know this is in progress, because we are met with a set of '{}' braces
  - -> if this hasn't worked, then these braces won't show
  - -> then he uses the .quit() method to close the session
  - -> this shows a message in the ipynb file, showing that the session is closed

- -> next, he is going to log into the browser to see if the email sent
- -> this allows us to send gmail emails from an ipynb file, as the email has now been successfully sent
- -> he has verified this in the browser
- -> this checking may be done automatically