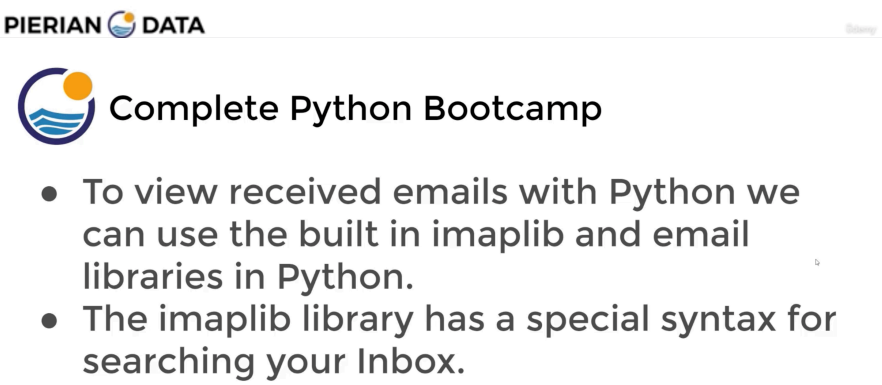
SECTION 18: EMAILS WITH PYTHON - 28 minutes, 3 parts

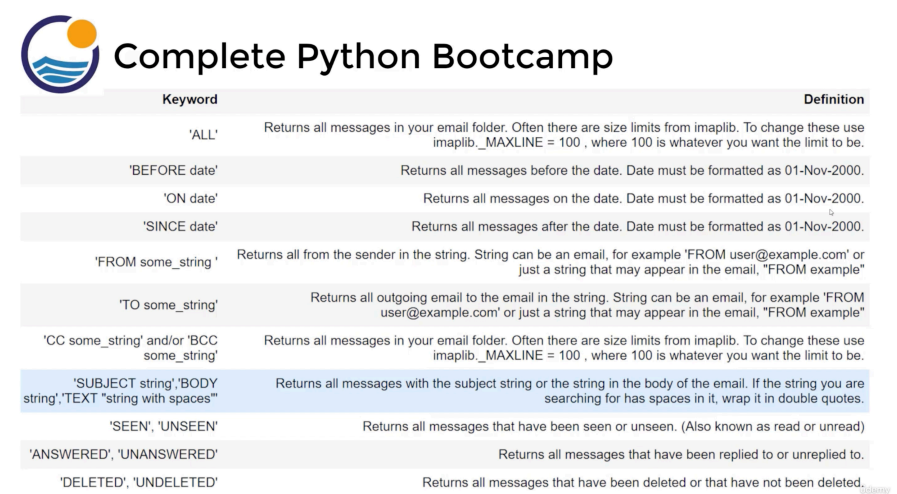
3/3 Receiving Emails with Python

- **-> receiving emails with Python**
  - -> exploring our emails with Python
  - -> this requires the imaplib and email libraries
  - -> imaplib has special syntax for searching an email inbox
  - -> this is equivalent to searching an email inbox
  - -> we can search for keywords in an email inbox
  - -> there are usually size limits to this
  - -> we can also edit their attributes
  - -> this can be changed to any number we want and implement size limits
  - -> this is used to filter emails, for example for things like dates etc
  - -> we can also search for emails from certain people / addresses, or with certain subjects / if the email has been deleted

Receiving Emails

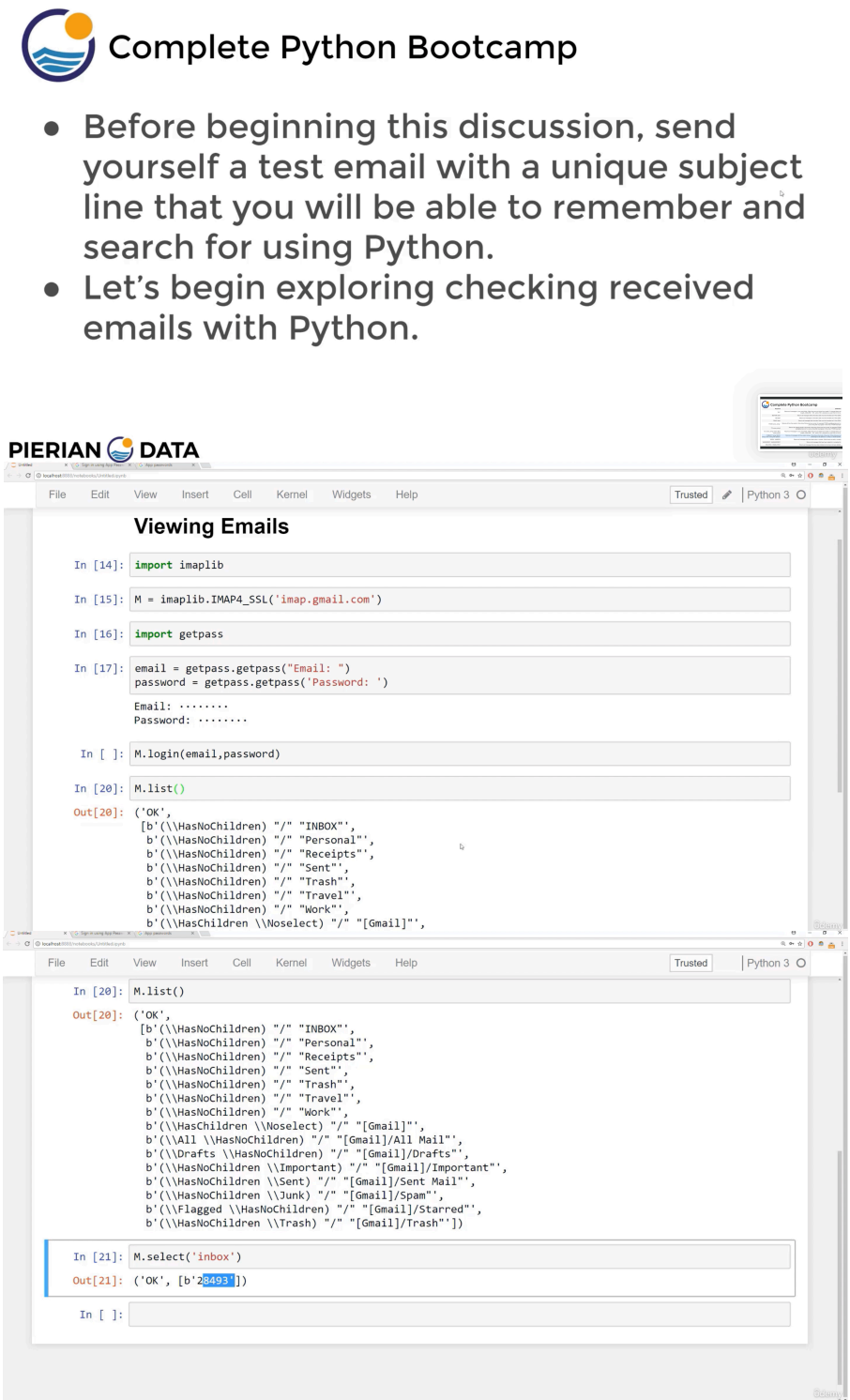


- **-> sending a test email**
  - -> we are sending ourself a test email, to receive using Python
  - -> he's doing this in an ipynb file now
  - -> first importing the imaplib module
  - -> next, he defines a variable called M using the IMAP4_SSL method
  - -> the argument to this is an email address using gmail
  - -> he then imports the getpass



Keyword	Definition
'ALL'	Returns all messages in your email folder. Often there are size limits from imaplib. To change these use imaplib._MAXLINE = 100 , where 100 is whatever you want the limit to be.
'BEFORE date'	Returns all messages before the date. Date must be formatted as 01-Nov-2000.
'ON date'	Returns all messages on the date. Date must be formatted as 01-Nov-2000.
'SINCE date'	Returns all messages after the date. Date must be formatted as 01-Nov-2000.
'FROM some_string '	Returns all from the sender in the string. String can be an email, for example 'FROM user@example.com' or just a string that may appear in the email, "FROM example"
'TO some_string'	Returns all outgoing email to the email in the string. String can be an email, for example 'FROM user@example.com' or just a string that may appear in the email, "FROM example"
'CC some_string' and/or 'BCC some_string'	Returns all messages in your email folder. Often there are size limits from imaplib. To change these use imaplib._MAXLINE = 100 , where 100 is whatever you want the limit to be.
'SUBJECT string','BODY string','TEXT "string with spaces"'	Returns all messages with the subject string or the string in the body of the email. If the string you are searching for has spaces in it, wrap it in double quotes.
'SEEN', 'UNSEEN'	Returns all messages that have been seen or unseen. (Also known as read or unread)
'ANSWERED', 'UNANSWERED'	Returns all messages that have been replied to or unreplied to.
'DELETED', 'UNDELETED'	Returns all messages that have been deleted or that have not been deleted.

module
- -> then he defines another variable called `email`, using the .getpass() method
- -> he repeats this process for a password
- -> running this cell asks the user firstly for an email and then a password
- -> because the .getpass() method was used for this, it means that when the user is asked for an input, what they type is entered in bullet points as with a password
- -> he then uses the .login() method to connect to the email and when he does this we are told in the Jupyter notebook that it's successful because a message appears below the cell
- -> M.list() can then be used to show information about different folders and flags in the gmail account which he just connected to in the ipynb file
- -> he then decides to target the inbox filter with this new cell
- -> he uses the .select() method to select for inbox
- **-> this returns 'OK' and a connection number**
  - -> this is how we know that we've connected successfully
  - -> this means that we've logged in and now want to search the inbox
  - -> there are multiple tags we can use to specify the message that we're searching for
- -> he defines two variables, typ and data - to implement tuple unpacking
- -> he sets this equal to a .search() method on the implemented with none and the search parameter for its argument to filter through the inbox
- -> we can also search from a certain string, or for example from a certain date
- -> we can also search by subject, which has to be entered in capital letters ('SUBJECT')
- -> he then runs this, and calls the values of the variable to check that this has worked
- -> this returns a unique ID which references the email to fetch the data
- -> he retrieves the first item in this list by using array syntax
- -> he sets the value of this equal to a variable

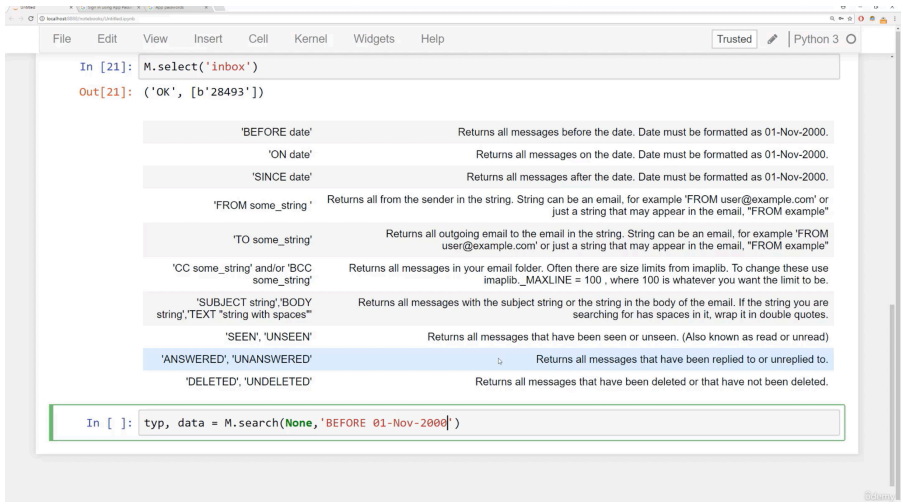


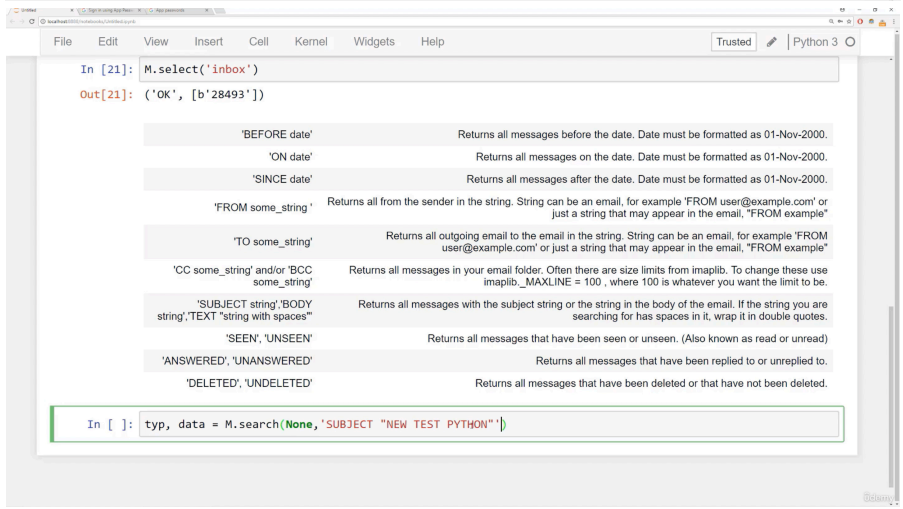
- **-> he then uses the .fetch() method to define two more variables**

- -> the arguments of this are the email id and a code which we are searching for
- -> he then prints out the contents for the email which he has just extracted, then comments it out
- -> he then extracts a raw email and stores it in a new variable
  - -> this is done using two square brackets next to each other, [][]
- -> he then defines another variable, using the .encode() method to decode this in utf-8 syntax
- -> then imports the email library, to extract the message from this string
- -> he then defines another variable called `email_message`, using the email_from_string method
- -> this gives us an email message object which is an iterator
- -> we can iterate through this email message, which he does in a for loop

- **-> this allows us to apply an if condition, while iterating through this**
  - -> he uses the .get_payload() method to print parts of this, assuming that they obey a given condition
  - -> this allows us to get rid of information which we don't want
  - -> some software engineers prefer not to use the email library, and write their own instead

- **-> he has extracted a raw email string from the email data**
  - -> then used an if block to grab the payload from this
  - -> this has been used in this example to print the content of an email





- **-> this is a content type which searches for plain text**
  - -> this can be changed to html, for the case that the user has provided a link to the email

- **-> summary**
  - -> he has imported IMAP lib
  - -> then connected to a gmail account
  - -> then given the user an option to enter in the email and password for the email to connect to
  - -> this is then logged into, and a list of options is returned as on the previous line
  - -> then using the .search() method, certain fields from this gmail account that we are logged into can be searched
  - -> once we have the emails of these, we then need to fetch them
  - -> the protocol for searching this has to be parsed in
  - -> once this is done, information from this email can be extracted