Automate The Boring Stuff Notes

Ch 2 - Flow Control

Mixing Boolean and Comparison Operators

- and, or, not same as &&, || in Java

“Truthy” and “Falsey” Values

When used in conditions, 0, 0.0, and " are considered False. Makes code easier to read

- ex. `while not name` is the same as `while not name != ''`
- ex. `if numOfGuests` is the same as `if numOfGuests != 0`

For Loop

- `for i in list` - same as foreach loop. Iterates through everything in the list
- `for i in range(len(list)):` iterate through indexes of list
- `for i in range(5)` - iterates from index 0 to 4
- `for i in range(12, 16)` - iterates from index 12 to 15 (last value is value right before n).
- `for i in range(0, 10, 2)` - iterates from 0 to 8, intervals of 2
- `for i in range(5, 0, -1):` - backwards iteration from 5 to 1

- `in, not` - is item in the list? is item not in the list?

range()

The return value from range(4) is a list-like value that Python considers similar to [0, 1, 2, 3]

- `for i in range(4)` is the same as `for i in [0, 1, 2, 3]:`

`i` represents the object of the list at that index. Like list[0], list [1]... `i` can be any variable name
First number in range is inclusive, second number exclusive

While loop

increment index i++ as `i += 1`

Exit program

`sys.exit()`

Ch 3 - Functions

None

`None` is the same as null
No return value in a function is the same as return None or return null. Like a void function in Java

print()

Can take multiple arguments, unlike Java

- `print('Hello')`
- `print('Hello', end='e')` - prints Helloe . adds e to end. if end not specified, newline character \n is added by default
- `print('cats', 'dogs', 'mice')` - prints 'cats dogs mice' with space as a separator
- `print('cats', 'dogs', 'mice', sep=',')` - prints 'cats,dogs,mice' with , as a separator

global

declare global variable with global

- i.e. `global x = 15`

Ch 4 - Lists

Negative indexes

First index from right to left starts at -1, -2, -3.. etc. Whereas first index from left to right starts at 0

- `spam = ['cat', 'bat', 'rat', 'elephant']`
- `spam[-1]` -- gives 'elephant'

Slice

Slice is inclusive of first number, exclusive of second number

- `spam` = ['cat', 'bat', 'rat', 'elephant']
- `spam[0:4]` - ['cat', 'bat', 'rat', 'elephant']
- `spam[1:3]` - 'bat', 'rat']
- `spam[0:-1]` - beginnning to index before last index - ['cat', 'bat', 'rat']
- `spam[:2]` - beginning to index 1 - ['cat', 'bat']
- `spam[1:]` - index 1 to the end - ['bat', 'rat', 'elephant']
- `spam[:]` - whole thing - ['cat', 'bat', 'rat', 'elephant']

len()

same as arr.length()

del

remove values from lists. like arr.remove(2)

- `del spam[2]` - remove rat

Multiple Assignment trick

- cat = `['fat', 'orange', 'loud']` --> returns size, color, disposition = cat

  same as `size = cat[0]`, `color = cat[1]`, and `disposition = cat[2]`

+=, *=, can do for ints, strings, and lists

Common list methods

- `list.append('value')`
- `list.insert (1, 'value')`
- `list.remove('value')` -- removes the first instance
- `list.sort()`
- `list.sort()(reverse=True)`
- `list.sort(key=sortcriteria)`

list (immutable) to tuple (mutable)

- `tuple(['cat', 'dog', 5])` - ('cat', 'dog', 5)
- `list(('cat', 'dog', 5))` - ['cat', 'dog', 5]
- `list('hello')` - ['h', 'e', 'l', 'l', 'o']

Copying a list:

- `copy.copy(list)` - copy a list to a new list, must import copy and call copy from copy
- `copy.deepcopy(list)` - copy a list to a new list if there are nested lists
- `list1 = list 2` - assigning a list to a new variable just assigns a reference to the list It doesnt copy a list

Ch 5 - Dictionaries

Dictionaries vs Lists

dictionaries are unordered so can't sort.
Integers can be used as keys for lists, but doesn't necessarily have to be

- `dict.keys()`
- `dict.values()`
- `dict.items()` - to get the key value pairs
- `if 'key' in dict.keys()` - to check if key is in dict
- `if 'values' in dict.values()` - to check if value is in dict
- `dict['key']` - access value of key
- `dict.get('key', 0))` - access value of key, if not then return default value of 0
- `dict.setdefault('key', 'value')` - add key/value pair to dict if key doesnt already have a value in the dict

Pretty Print list

`import pprint, pprint.pprint(list)`

Ch 6 - Strings

Escape characters in string

- `\t` - Tab
- `\n` - Newline (line break)
- `\\` - Backslash

in, not

`in`, `not` - are chars in the string? are chars not in the string? Like contains in Java

Printing techniques for a string

- `print(r'That is Carol\'s cat.')` - r at the start of print, keeps the string as a raw string
- `'''` - at the start of print, preserves all the newlines

```
print('''Dear Alice,

Eve's cat has been arrested for catnapping, cat burglary, and extortion.

Sincerely,
Bob''')
```

String Methods

- `str.upper()` - same as Java str.toUpperCase()
- `str.lower()` - same as Java str.toLowerCase()
- `str.isupper()` - true if all upper
- `str.islower()` - true if all lower
- `str.isalpha()` - true if all alphabet
- `str.isalnum()` - true if all alphanumeric
- `str.isdecimal()` - true if all numeric
- `str.isspace()` - true if only of spaces, tabs, and new-lines

- `istitle()` - true if only has words that begin with an uppercase letter followed by only lowercase letters.
- `'Hello world!'.startswith('Hello')`
- `'Hello world!'.endswith('world!')`

Join strings

- `'separator'.join(['a', 'b', 'c'])` - Join into string
  - `' '.join(['My', 'name', 'is', 'Simon'])` --> 'My name is Simon'

Split a string

- `str.split(separator).` - Split into an array. If no separator, then split on the space
  - `'My name is Simon'.split('m')` ---> ['My na', 'e is Si', 'on']
- `str.split('\n')` - split on newline
- `str.rjust(numOfCharacters, paddedCharacter)`
- `str.ljust(numOfCharacters, paddedCharacter)`
- `str.center(numOfCharacters, paddedCharacter)` - returns a padded version of the string they are called on

Strip a string of whitespace

If a string was `str = '    Hello World     '`, we can do

- `str.strip()`
- `str.lstrip()`
- `str.rstrip()`

Copy and paste a string

pyperclip module has copy() and paste() functions that can send text to and receive text from your computer's clipboard

- `import pyperclip`
- `pyperclip.copy('Hello world!')`
- `pyperclip.paste()`

Text is a common form of data, and Python comes with many helpful string methods to process the text stored in string values. Will make use of indexing, slicing, and string methods in almost every Python program we write

Ch 7 - Pattern Matching with Regular Expressions

Some Syntax

`\d` - digit
`({n})` - match this pattern n number of times

Example of a phone number

```
\d\d\d-\d\d\d-\d\d\d\d
\d{3}-\d{3}-\d{4}
```

How to use Regex

1. Import the regex module with import re.
2. Create a Regex object with the re.compile() function. (Remember to use a raw string.)
3. Pass the string you want to search into the Regex object's search() method. This returns a Match object.
4. Call the Match object's group() method to return a string of the actual matched text.

Example

- `phoneNumRegex = re.compile(r'\d\d\d-\d\d\d-\d\d\d\d')`
- `mo = phoneNumRegex.search('My number is 415-555-4242.')`
- `print('Phone number found: ' + mo.group())`

Grouping, matching text

Adding parentheses to create different groups

- `(\d\d\d)-(\d\d\d-\d\d\d\d)`

group(num) to match different parts of the matched text surrounded by the parentheses

- `phoneNumRegex = re.compile(r'(\d\d\d)-(\d\d\d-\d\d\d\d)')`
- `mo = phoneNumRegex.search('My number is 415-555-4242.')`
- `mo.group(1)` - '415'
- `mo.group(2)` - '555-4242'
- `mo.group(0)` - '415-555-4242'
- `mo.group()` - '415-555-4242'

groups() to get all the groups. Can do multiple assignment variables

- `mo.groups()` - ('415', '555-4242')
- `areaCode, mainNumber = mo.groups()`
- `print(areaCode)` - 415
- `print(mainNumber)` - 555-4242

if we need to match parentheses in the text, need to escape the (and) characters with a backslash

- `phoneNumRegex = re.compile(r'(\(\d\d\d\)) (\d\d\d-\d\d\d\d)')`
- `mo = phoneNumRegex.search('My phone number is (415) 555-4242.')`
- `mo.group(1) - '(415)'
- `mo.group(2) - '555-4242'

`r'Batman|Tina Fey'` , use the pipe | to match either of the expressions on the pipe. When both occurs, you match the first one that occurs

- `heroRegex = re.compile (r'Batman|Tina Fey')`
- `mo1 = heroRegex.search('Batman and Tina Fey.')`
- `mo1.group()` - 'Batman'
- `mo2 = heroRegex.search('Tina Fey and Batman.')`
- `mo2.group()` - 'Tina Fey

Can use the pipe to match one of several patterns as part of regex, using parentheses. I.e. match the prefix, match the suffix

- `batRegex = re.compile(r'Bat(man|mobile|copter|bat)')`
- `mo = batRegex.search('Batmobile lost a wheel')`
- `mo.group()`- 'Batmobile'
- `mo.group(1)` - 'mobile'

The `?` matches zero or one of the preceding parentheses group
i.e. Everything in parenthesis before question mark is optional part of pattern

- `phoneRegex = re.compile(r'(\d\d\d-)?\d\d\d-\d\d\d\d')`
- `mo1 = phoneRegex.search('My number is 415-555-4242')`
- `mo1.group()` - '415-555-4242'
- `mo2 = phoneRegex.search('My number is 555-4242')`
- `mo2.group()` - '555-4242'

The `*` matches zero or more of the preceding group.
i.e. Everything in parenthesis before star is optional part of pattern and can be repeated multiple times in pattern match

- `batRegex = re.compile(r'Bat(wo)*man')`
- `mo1 = batRegex.search('The Adventures of Batman')`
- `mo1.group()` - 'Batman'
- `mo2 = batRegex.search('The Adventures of Batwoman')`
- `mo2.group()` - 'Batwoman'
- `mo3 = batRegex.search('The Adventures of Batwowowowoman')`
- `mo3.group()` - 'Batwowowowoman'

The `+` matches one or more of the preceding group.
The `{n}` matches exactly n of the preceding group.
The `{n,}` matches n or more of the preceding group.
The `{,m}` matches 0 to m of the preceding group.
The `{n,m}` matches at least n and at most m of the preceding group.
`{n,m}?` or `*?` or `+?` performs a nongreedy match of the preceding group.

i.e. Match 3, 4, or 5 instances of string. Matches the longest string first i.e. HaHaHaHaHA

- `greedyHaRegex = re.compile(r'(Ha){3,5}')`

To match the shortest string first, use {3,5}?
i.e. match a specific number of repetitions

- `haRegex = re.compile(r'(Ha){3}')`
- `mo1 = haRegex.search('HaHaHa')`
- `mo1.group()` - 'HaHaHa'

`search()` will return a Match object of the first matched text in the searched string

`findall()` method will return the strings of every match in the searched string

`^spam` means the string must begin with spam.
`spam$` means the string must end with spam.

- `beginsWithHello = re.compile(r'^Hello')`
- `beginsWithHello.search('Hello world!')`

The `.` matches any character, except newline characters.

- `atRegex = re.compile(r'.at')`
- `atRegex.findall('The cat in the hat sat on the flat mat.')` - ['cat', 'hat', 'sat', 'lat', 'mat']

`\d`, `\w`, and `\s` match a digit, word, or space character, respectively.
`\D`, `\W`, and `\S` match anything except a digit, word, or space character, respectively.

Can chain rules together

The `r'^\d+$'` regular expression string matches strings that both begin and end with one or more numeric characters.

- `wholeStringIsNum = re.compile(r'^\d+$')`

Define own character classes

`[abc]` matches any character between the brackets (such as a, b, or c).
`[^abc]` matches any character that isn't between the brackets.

- `vowelRegex = re.compile(r'[aeiouAEIOU]')`
- `vowelRegex.findall('Robocop eats baby food. BABY FOOD.')` - ['o', 'o', 'o', 'e', 'a', 'a', 'o', 'o', 'A', 'O', 'O']

Make regex case-insensitive

pass re.IGNORECASE or re.I as a second argument to re.compile().
i.e. `robocop = re.compile(r'robocop', re.I)`

Substituting Strings in regex

- `namesRegex = re.compile(r'Agent \w+')`
- `namesRegex.sub('CENSORED', 'Agent Alice gave the secret documents to Agent Bob.')` - 'CENSORED gave the secret documents to CENSORED.'

Spread regex over multiple lines

Do so with comments like this:

```
phoneRegex = re.compile(r'''(
    (\d{3}|\(\d{3}\))?            # area code
    (\s|-|\.)?                    # separator
    \d{3}                         # first 3 digits
    (\s|-|\.)                     # separator
    \d{4}                         # last 4 digits
    (\s*(ext|x|ext.)\s*\d{2,5})?  # extension
    )''', re.VERBOSE)
```

Ch 8 - Reading / Writing Files

Generate a file path

- `os.path.join('usr', 'bin', 'spam')` - returns usr/bin/spam'. (will do / if Windows, \ if Mac)
- `os.chdir('C:\\Windows\\System32')`
- `os.getcwd()`
- `os.makedirs('C:\\delicious\\walnut\\waffles')`
- `path = 'C:\\Windows\\System32\\calc.exe'`
- `os.path.basename(path)` - 'calc.exe'
- `os.path.dirname(path)` - 'C:\Windows\System32'

Get paths

- `os.path.abspath('.')` - 'C:\Python34'
- `os.path.abspath('.\\Scripts')` - 'C:\Python34\Scripts'
- `os.path.isabs('.')`
- `os.path.relpath('C:\\Windows', 'C:\\')` - 'Windows'

Splitting on path returns a tuple with the base name and dir name

- `calcFilePath = 'C:\\Windows\\System32\\calc.exe'`
- `os.path.split(calcFilePath)` - ('C:\Windows\System32', 'calc.exe')

If we want split as a list of strings use os.path.sep. First list value is a blank for Mac

- `'/usr/bin'.split(os.path.sep)` - ['', 'usr', 'bin']
- `os.path.getsize('C:\\Windows\\System32\\calc.exe')`

Returns a list of directory items

- `os.listdir('C:\\Windows\\System32')`
- `os.path.exists('C:\\some_made_up_folder')`
- `os.path.isdir('C:\\Windows\\System32')`
- `os.path.isfile('C:\\Windows\\System32')`

Reading and writing files in Python

There are three steps to reading or writing files in Python.

1. Call the open() function to return a File object.
2. Call the read() or write() method on the File object.
3. Close the file by calling the close() method on the File object.

- `helloFile = open('/Users/your_home_folder/hello.txt')`

Returns a long string of everything being read

- `helloContent = helloFile.read()`

Returns a list of string values that ends in a newline character for each string value being read

- `sonnetFile.readlines()`

W is write, a is append

- `baconFile = open('bacon.txt', 'w')`
- `baconFile.write('Hello world!\n')`
- `baconFile = open('bacon.txt', 'a')`

Saving variables

Can save variables in your Python programs to binary shelf files using the shelve module.

- `import shelve`
- `shelfFile = shelve.open('mydata')`
- `cats = ['Zophie', 'Pooka', 'Simon']`
- `shelfFile['cats'] = cats`

Pretty Printing and Formatting

`pprint.pprint()` function will “pretty print” the contents of a list or dictionary to the screen

`pprint.pformat()` function will return this same text as a string instead of printing it.

ex. have dictionary stored in a variable and want to save this variable and its contents for future use. pprint.pformat() gives a string that you can write to .py file.

- `import pprint`
- `cats = [{'name': 'Zophie', 'desc': 'chubby'}, {'name': 'Pooka', 'desc': 'fluffy'}]`
- `pprint.pformat(cats)` - "[{'desc': 'chubby', 'name': 'Zophie'}, {'desc': 'fluffy', 'name': 'Pooka'}]"
- `fileObj = open('myCats.py', 'w')`
- `fileObj.write('cats = ' + pprint.pformat(cats) + '\n')`

Ch 9 – Organizing Files

shutil

shutil (or shell utilities) module - copy, move, rename, and delete files

Copy files

- `shutil.copy(source, destination)` - returns a string of the path of the copied file.

Copytree - copy an entire folder and every folder and file contained in it

- `shutil.copytree(source, destination)`

Move files

- `shutil.move(source, destination)`

Return a string of the absolute path of the new location.

If there’s a file with that name in the destination folder already, it will be overwritten

If the destination folder doesn’t exist, the file gets renamed to the name of the desired destination folder, which is not what we want.

Delete files and folders.

- `os.unlink(path)` - delete the file at path.
- `os.rmdir(path)` - delete an empty folder at path.
- `shutil.rmtree(path)` - remove the folder at path, and all files and folders it contains will also be deleted.

These methods delete the files ENTIRELY AND CANNOT BE UNDONE.

Safe Deletes with the send2trash Module

- `import send2trash`
- `send2trash.send2trash('bacon.txt')`

Walking Through a Directory Tree and touching each file as we go

`os.walk()` function will return three values on each iteration through the loop:

1. A string of the current folder's name
2. A list of strings of the folders in the current folder
3. A list of strings of the files in the current folder

Example

```
for folderName, subfolders, filenames in os.walk('C:\\delicious'):
    print('The current folder is ' + folderName)

    for subfolder in subfolders:
        print('SUBFOLDER OF ' + folderName + ': ' + subfolder)
    for filename in filenames:
        print('FILE INSIDE ' + folderName + ': '+ filename)

    print('')
```

Zip Files

Python programs can both create and open (or extract) ZIP files using functions in the zipfile module.

Read in an existing .zip file

Call the `zipfile.ZipFile()` function

- `import zipfile, os`
- `os.chdir('C:\\')` - move to the folder with example.zip
- `exampleZip = zipfile.ZipFile('example.zip')`

List of items in zipFile

- `exampleZip.namelist()` - list of items in zipfile. ['spam.txt', 'cats/', 'cats/catnames.txt', 'cats/zophie.jpg']

Extract the zipfile (to current folder or another folder)

- `exampleZip.extractall()`
- `exampleZip.extract('spam.txt')`
- `exampleZip.extract('spam.txt', 'C:\\some\\new\\folders')`
- `exampleZip.close()`

Create and Add to Zip file

- `import zipfile`
- `newZip = zipfile.ZipFile('new.zip', 'w')`
- `newZip.write('spam.txt', compress_type=zipfile.ZIP_DEFLATED)`
- `newZip.close()`

Ch 10 – Debugging

Try catch

Assert statements

Using the logging module

Ch 11 – Web Scraping

Modules

Webbrowser. Opens a browser to a specific page, included with python

Requests. Downloads files and web pages from the Internet.

Beautiful Soup. Parses HTML

Selenium. Launches and controls a web browser. Selenium is able to fill in forms and simulate mouse clicks in this browser.

Webbrowser - open a web browser

- `import webbrowser`
- `webbrowser.open('http://inventwithpython.com/')`

Requests – download files/web pages from internet

- `requests.get()` – downloads a web page. Returns a Response object
- `import requests`
- `res = requests.get('https://automatetheboringstuff.com/files/rj.txt')`

Catch exceptions if webpage is unable to be downloaded

```
try:
    res.raise_for_status()
except Exception as exc:
    print('There was a problem: %s' % (exc))
```

Download and Save a File

Call `requests.get()` to download the file.

Call `open()` with 'wb' to create a new file in write binary mode. B for binary

Loop over the Response object’s iter_content() method.

Call `write()` on each iteration to write the content to the file.

Call `close()` to close the file.

- `import requests`
- `res = requests.get('https://automatetheboringstuff.com/files/rj.txt')`
- `res.raise_for_status()`
- `playFile = open('RomeoAndJuliet.txt', 'wb')`
-

```
for chunk in res.iter_content(100000):
        playFile.write(chunk)
```

- `playFile.close()`

Beautiful Soup – to parse HTML

Using the Developer Tools to Find HTML Elements
Creating a BeautifulSoup Object from HTML

Load a website into a Beautiful Soup Object

- `import requests, bs4`
- `res = requests.get('http://nostarch.com')``
- `noStarchSoup = bs4.BeautifulSoup(res.text)

Finding an Element with the `select()` Method

return a list of Tag objects, which is how Beautiful Soup represents an HTML element.
one Tag object for every match in the BeautifulSoup object's HTML
Tag values can be passed to the `str()` function to show the HTML tags they represent
Tag values also have an attrsattribute that shows all the HTML attributes of the tag as a dictionary.

- `soup.select('div')` - All elements named <div>
- `soup.select('#author')` - The element with an id attribute of author
- `soup.select('.notice')` - All elements that use a CSS class attribute named notice
- `soup.select('div span')` - All elements named <span> that are within an element named <div>
- `soup.select('div > span')` - All elements named <span> that are directly within an element named <div>, with no other element in between
- `soup.select('input[name]')` - All elements named that have a name attribute with any value
- `soup.select('input[type="button"]')` - All elements named that have an attribute named type with value button
- `soup.select('p #author')` – all elements that has an idattribute of author, as long as it is also inside a element.

Example

- `elems = soup.select('#author')`
- `elems [0].getText()` - 'Al Sweigart'
- `str(elems [0])` - 'Al Sweigart'
- `elems [0].attrs` - {'id': 'author'}

Selenium - to control the Web Browser

```
from selenium import webdriver
browser = webdriver.Firefox()
browser.get('http://inventwithpython.com')
```

Use WebDriver Methods for Finding Elements. Surround with try/catch block. Returns a WebElement object or list

- `browser.find_element_by_class_name(name)`
- `browser.find_elements_by_class_name(name)`
- `browser.find_element_by_css_selector(selector)`
- `browser.find_elements_by_css_selector(selector)`
- `browser.find_element_by_id(id)`
- `browser.find_elements_by_id(id)`
- `browser.find_element_by_link_text(text)`
- `browser.find_elements_by_link_text(text)`
- `browser.find_element_by_partial_link_text(text)`
- `browser.find_elements_by_partial_link_text(text)`
- `browser.find_element_by_name(name)`
- `browser.find_elements_by_name(name)`
- `browser.find_element_by_tag_name(name)`
- `browser.find_elements_by_tag_name(name)`

Once you have the WebElement object, you can find out more about it by reading the attributes or calling the methods

- `tag_name` - The tag name, such as 'a' for an <a> element
- `get_attribute(name)` - The value for the element’s name attribute
- `text` - The text within the element, such as 'hello' in hello
- `clear()` - For text field or text area elements, clears the text typed into it
- `is_displayed()` - Returns True if the element is visible; otherwise returns False
- `is_enabled()` - For input elements, returns True if the element is enabled; otherwise returns False
- `is_selected()` - For checkbox or radio button elements, returns True if the element is selected; otherwise returns False
- `location` - A dictionary with keys 'x' and 'y' for the position of the element in the page

ex.

```
from selenium import webdriver
browser = webdriver.Firefox()
browser.get('http://inventwithpython.com')
try:
    elem = browser.find_element_by_class_name('bookcover')
    print('Found <%s> element with that class name!' % (elem.tag_name))
except:
    print('Was not able to find an element with that name.')
```

This program will output the following:

Found element with that class name!

Click an element

- `elem.click()`

Fill out a form: find the input or textarea element

- `passwordElem.send_keys('12345')`
- `passwordElem.submit()`

Send keyboard keys

- `Keys.DOWN, Keys.UP, Keys.LEFT, Keys.RIGHT` - The keyboard arrow keys
- `Keys.ENTER, Keys.RETURN - The ENTER and RETURN keys`
- `Keys.HOME, Keys.END, Keys.PAGE_DOWN, Keys.PAGE_UP` - The home, end, pagedown, and pageup keys
- `Keys.ESCAPE, Keys.BACK_SPACE, Keys.DELETE` - The ESC, BACKSPACE, and DELETE keys
- `Keys.F1, Keys.F2,..., Keys.F12` - The F1 to F12 keys at the top of the keyboard
- `Keys.TAB` - The TAB key

Example

- `htmlElem = browser.find_element_by_tag_name('html')`
- `htmlElem.send_keys(Keys.END)     # scrolls to bottom`
- `htmlElem.send_keys(Keys.HOME)    # scrolls to top`

Click browser buttons

- `browser.back()` - Clicks the Back button.
- `browser.forward()` - Clicks the Forward button.
- `browser.refresh()` - Clicks the Refresh/Reload button.
- `browser.quit()` - Clicks the Close Window button.

Ch 12 – Working with Excel Spreadsheets

How to read cells out of spreadsheet file

1. Import the openpyxl module.
2. Call the openpyxl.load_workbook() function.
3. Get a Workbook object.
4. Read the active member variable or call the get_sheet_by_name() workbook method.
5. Get a Worksheet object.
6. Use indexing or the cell() sheet method with row and column keyword arguments.
7. Get a Cell object.
8. Read the Cell object’s value attribute.

Open excel file / workbook

- `import openpyxl`
- `wb = openpyxl.load_workbook('example.xlsx')`

Get Sheets

- `wb.get_sheet_names()`- ['Sheet1', 'Sheet2', 'Sheet3']
- `sheet = wb.get_sheet_by_name('Sheet3')`
  - `sheet` - <Worksheet "Sheet3">
- `sheet.title` - 'Sheet3'
- `anotherSheet = wb.active`
  - `anotherSheet` - <Worksheet "Sheet1">

Get Cells from Sheets

- `sheet = wb.get_sheet_by_name('Sheet1')`
  - `sheet['A1']` - <Cell Sheet1.A1>
  - `sheet['A1'].value` - datetime.datetime(2015, 4, 5, 13, 34, 2)
  - `c = sheet['B1']`
    - `c.value` - 'Apples'
    - `'Row ' + str(c.row) + ', Column ' + c.column + ' is ' + c.value` - 'Row 1, Column B is Apples'
    - `'Cell ' + c.coordinate + ' is ' + c.value` - 'Cell B1 is Apples'

get a cell using the sheet’s cell() method

Specifying a column by letter can be tricky to program, especially because after column Z, the columns start by using two letters: AA, AB, AC, and so on.

As an alternative, you can also get a cell using the sheet’s cell() method and passing integers for its row and column keyword arguments.

The first row or column integer is 1, not 0.

- `sheet.cell(row=1, column=2)` - <Cell Sheet1.B1>
- `sheet.cell(row=1, column=2).value` - 'Apples'
-

```
for i in range(1, 8, 2):
        print(i, sheet.cell(row=i, column=2).value)
```

1 Apples

3 Pears

5 Apples

7 Strawberries

Max Row and Column

- `sheet.max_row` - 7
- `sheet.max_column` - 3

column returns as a number, not a letter like in excel

Converting Between Column Letters and Numbers

- `import openpyxl`
- `from openpyxl.cell import get_column_letter, column_index_from_string`
- `get_column_letter(27)` - 'AA'
- `column_index_from_string('AA')` - 27

Getting Rows and Columns from the Sheets

Can slice Worksheet objects to get all the Cell objects in a row, column, or rectangular area. Then can loop over all the cells in the slice. Like rectangular select in Excel

- `import openpyxl` -
- `wb = openpyxl.load_workbook('example.xlsx')`
- `sheet = wb.get_sheet_by_name('Sheet1')`
- `tuple(sheet['A1':'C3'])` - ((<Cell Sheet1.A1>, <Cell Sheet1.B1>, <Cell Sheet1.C1>), (<Cell Sheet1.A2>, <Cell Sheet1.B2>, <Cell Sheet1.C2>), (<Cell Sheet1.A3>, <Cell Sheet1.B3>, <Cell Sheet1.C3>))
-

```
for rowOfCellObjects in sheet['A1':'C3']:
        for cellObj in rowOfCellObjects:
               print(cellObj.coordinate, cellObj.value)
           print('--- END OF ROW ---')
```

For one column or one row, can do something like this

- `sheet.columns[1]` - (<Cell Sheet1.B1>, <Cell Sheet1.B2>, <Cell Sheet1.B3>, <Cell Sheet1.B4>, <Cell Sheet1.B5>, <Cell Sheet1.B6>, <Cell Sheet1.B7>)
-

```
for cellObj in sheet.columns[1]:
        print(cellObj.value)
```

Creating and Saving Excel Documents

New blank workbook object

- `wb = openpyxl.Workbook()`

First sheet default name is sheet, so we can rename the sheet.title

- `wb.get_sheet_names()` - ['Sheet']
- `sheet = wb.active`
- `sheet.title` - 'Sheet'
- `sheet.title = 'Spam Bacon Eggs Sheet'`

Save workbook

- `wb.save('example_copy.xlsx')`

Create and remove sheets

- `wb.create_sheet()`
- `wb.create_sheet(index=0, title='First Sheet')` - moves new sheet to index 0 position
- `wb.remove_sheet(wb.get_sheet_by_name('Middle Sheet'))`
- `wb.remove_sheet(wb.get_sheet_by_name('Sheet1'))`

Write values to cells / update sheet

- `sheet['A1'] = 'Hello world!'`
- `sheet['A1'].value` - 'Hello world!'

Set font. Can set name, size, fold, italic

- `fontObj1 = Font(name='Times New Roman', bold=True)`
- `italic24Font = Font(size=24, italic=True)`
- `sheet['A1'].font = italic24Font`

Formulas

- `sheet['B9'] = '=SUM(B1:B8)'` - This will store =SUM(B1:B8) as the value in cell B9. This sets the B9 cell to a formula that calculates the sum of values in cells B1 to B8.

Adjusting Rows and Columns

Setting Row Height and Column Width

- `sheet.row_dimensions[1].height = 70`
- `sheet.column_dimensions['B'].width = 20`

Merging and Unmerging Cells

- `sheet.merge_cells('A1:D3')`
- `sheet.unmerge_cells('A1:D3')`

Freeze panes

- `sheet.freeze_panes = 'A2'`

If you set the freeze_panes attribute to 'A2', row 1 will always be viewable, no matter where the user scrolls in the spreadsheet. Freeze pane is set to what the first unfrozen pane will be

Unfreeze panes

- `sheet.freeze_panes = 'A1'` or
- `sheet.freeze_panes = None`

Create a chart – bar/line/scatter/pie chart

1. Create a Reference object from a rectangular selection of cells.
2. Create a Series object by passing in the Reference object.
3. Create a Chart object.
4. Append the Series object to the Chart object.
5. Add the Chart object to the Worksheet object, optionally specifying which cell the top left corner of the chart should be positioned..

Reference object

Reference objects are created by calling the openpyxl.chart.Reference() function and passing three arguments:

1. The Worksheet object containing your chart data.
2. A tuple of two integers, representing the top-left cell of the rectangular selection of cells containing your chart data: The first integer in the tuple is the row, and the second is the column. Note that 1 is the first row, not 0.
3. A tuple of two integers, representing the bottom-right cell of the rectangular selection of cells containing your chart data: The first integer in the tuple is the row, and the second is the column.

Example

- `import openpyxl`
- `wb = openpyxl.Workbook()`
- `sheet = wb.active`
-

```
for i in range(1, 11):         # create some data in column A
        sheet['A' + str(i)] = i
```

- `refObj = openpyxl.chart.Reference(sheet, min_col=1, min_row=1, max_col=1, max_row=10)`
- `seriesObj = openpyxl.chart.Series(refObj, title='First series')`
- `chartObj = openpyxl.chart.BarChart()`
- `chartObj.title = 'My Chart'`
- `chartObj.append(seriesObj)`
- `sheet.add_chart(chartObj, 'C5')`
- `wb.save('sampleChart.xlsx')`

Ch 13 – Working with PDF and Word Documents

Import PyPDF2 and Python-Docx for PDF and Word Documents

Extract text from PDF

- `import PyPDF2`
- `pdfFileObj = open('meetingminutes.pdf', 'rb')` - rb is read binary. Wb is write binary
- `pdfReader = PyPDF2.PdfFileReader(pdfFileObj)`
- `pdfReader.numPages` - 19
- `pageObj = pdfReader.getPage(0)`
- `pageObj.extractText()` - text extraction isn't perfect

Decrypting PDFs

- `pdfReader = PyPDF2.PdfFileReader(open('encrypted.pdf', 'rb'))`
- `pdfReader.isEncrypted` - True
- `pdfReader.decrypt('password')`
- `pdfReader.getPage(0)`

Creating PDFs

PyPDF2 doesn't allow you to directly edit a PDF. Instead, you have to create a new PDF and then copy content over from an existing document

1. Open one or more existing PDFs (the source PDFs) into PdfFileReader objects.
2. Create a new PdfFileWriter object.
3. Copy pages from the PdfFileReader objects into the PdfFileWriter object.
4. Finally, use the PdfFileWriter object to write the PdfFileWriter object to the output PDF

Copying pages also allows you to combine multiple PDF files, cut unwanted pages, or reorder pages.

Example

- `import PyPDF2`
- `pdf1File = open('meetingminutes.pdf', 'rb')`
- `pdf2File = open('meetingminutes2.pdf', 'rb')`
- `pdf1Reader = PyPDF2.PdfFileReader(pdf1File)`
- `pdf2Reader = PyPDF2.PdfFileReader(pdf2File)`
- `pdfWriter = PyPDF2.PdfFileWriter()`
-
  ```
  for pageNum in range(pdf1Reader.numPages):
          pageObj = pdf1Reader.getPage(pageNum)
          pdfWriter.addPage(pageObj)
  ```
-
  ```
  for pageNum in range(pdf2Reader.numPages):
          pageObj = pdf2Reader.getPage(pageNum)
          pdfWriter.addPage(pageObj)
  ```
- `pdfOutputFile = open('combinedminutes.pdf', 'wb')`
- `pdfWriter.write(pdfOutputFile)`
- `pdfOutputFile.close()`
- `pdf1File.close()`
- `pdf2File.close()`

Rotating Pages

- `page = pdfReader.getPage(0)`
- `page.rotateClockwise(90)`
- `page.rotateCounterclockwise(90)`

Overlaying pages

Overlay the contents of one page over another, useful for adding a logo, timestamp, or watermark to a page

Call a Page object on one page, then call a call mergePage() with a page from another doc. Add the pages of the remaining PDF and then write out the PDF

- `import PyPDF2`
- `minutesFile = open('meetingminutes.pdf', 'rb')`
- `pdfReader = PyPDF2.PdfFileReader(minutesFile)`
- `minutesFirstPage = pdfReader.getPage(0)`
- `pdfWatermarkReader = PyPDF2.PdfFileReader(open('watermark.pdf', 'rb'))`
- `minutesFirstPage.mergePage(pdfWatermarkReader.getPage(0))`
- `pdfWriter = PyPDF2.PdfFileWriter()`
- `pdfWriter.addPage(minutesFirstPage)`
-

```
for pageNum in range(1, pdfReader.numPages):
            pageObj = pdfReader.getPage(pageNum)
            pdfWriter.addPage(pageObj)
```

- `resultPdfFile = open('watermarkedCover.pdf', 'wb')`
- `pdfWriter.write(resultPdfFile)`
- `minutesFile.close()`
- `resultPdfFile.close()`

Run object

A Run object is a contiguous run of text with the same style. A new Run object is needed whenever the text style changes.

Encrypting PDFs

- `pdfWriter.encrypt('password')`

Reading Word Documents

- `import docx`
- `doc = docx.Document('demo.docx')`

Get full text from a .docx File

```
fullText = []
for para in doc.paragraphs:
    fullText.append(para.text)
print('\n'.join(fullText))
```

join strings of the paragraphs together with newline characters and print it out

Style attributes for runs

- 'Normal'
- 'BodyText'
- 'BodyText2'
- 'BodyText3'

- 'Caption'
- 'Heading1'
- 'Heading2'
- 'Heading3'
- 'Heading4'
- 'Heading5'
- 'Heading6'
- 'Heading7'
- 'Heading8'
- 'Heading9'
- 'IntenseQuote'
- 'List'
- 'List2'
- 'List3'
- 'ListBullet'
- 'ListBullet2'
- 'ListBullet3'
- 'ListContinue'
- 'ListContinue2'
- 'ListContinue3'
- 'ListNumber'
- 'ListNumber2'
- 'ListNumber3'

Text attributes for runs

- bold
- italic
- underline
- strike
- double_strike
- all_caps
- small_caps
- shadow - The text appears with a shadow.
- Outline - The text appears outlined rather than solid.
- rtl
- imprint - The text appears pressed into the page.
- Emboss - The text appears raised off the page in relief.

Example of changing style and text attributes on runs

- `doc = docx.Document('demo.docx')`
- `doc.paragraphs[0].text` - 'Document Title'
- `doc.paragraphs[0].style` - 'Title'
- `doc.paragraphs[0].style = 'Normal'`
- `doc.paragraphs[1].text` - 'A plain paragraph with some bold and some italic'
- `(doc.paragraphs[1].runs[0].text, doc.paragraphs[1].runs[1].text, doc.paragraphs[1].runs[2].text, doc.paragraphs[1].runs[3].text)` - ('A plain paragraph with some ', 'bold', ' and some ', 'italic')
- `doc.paragraphs[1].runs[0].style = 'QuoteChar'`
- `doc.paragraphs[1].runs[1].underline = True`
- `doc.paragraphs[1].runs[3].underline = True`

Write Documents

- `import docx`
- `doc = docx.Document()`
- `doc.add_paragraph('Hello world!')`
- `doc.save('helloworld.docx')`

Add Heading

integer 0 makes the heading the Title style for the top of the document. Integers 1 to 4 are for various heading levels, with 1 being the main heading and 4 the lowest subheading

- `doc.add_heading('Header 0', 0)`

Add Line break

- `doc.add_paragraph('This is on the first page!')`
- `doc.paragraphs[0].runs[0].add_break(docx.text.WD_BREAK.PAGE)`

Add picture

- `doc.add_picture('zophie.png', width=docx.shared.Inches(1), height=docx.shared.Cm(4))`

Ch 14 – Working with CSV Files and JSON Data

Read CSV

- `import csv`
- `exampleFile = open('example.csv')`
- `exampleReader = csv.reader(exampleFile)`
- `exampleData = list(exampleReader)`
- `exampleData -` [['4/5/2015 13:34', 'Apples', '73'], ['4/5/2015 3:41', 'Cherries', '85'], ['4/6/2015 12:46', 'Pears', '14'], ['4/8/2015 8:59', 'Oranges', '52'], ['4/10/2015 2:07', 'Apples', '152'], ['4/10/2015 18:10', 'Bananas', '23'], ['4/10/2015 2:40', 'Strawberries', '98']]

CSV is a list of lists as exampleData[row][col]

Write CSV

- `import csv`
- `outputFile = open('output.csv', 'w', newline='') -` must include the `newline=''` to prevent double spacing
- `outputWriter = csv.writer(outputFile)`
- `outputWriter.writerow(['spam', 'eggs', 'bacon', 'ham'])-` 21
- `outputFile.close()

Keyword args for CSV writer

Delimiter – delimit row values by something other than comma

Line terminator – end line with something other than newline

- `csvWriter = csv.writer(csvFile, delimiter='\t', lineterminator='\n\n')`

Read JSON with JSON.loads()

- `stringOfJsonData = '{"name": "Zophie", "isCat": true, "miceCaught": 0, "felineIQ": null}'`
- `import json`
- `jsonDataAsPythonValue = json.loads(stringOfJsonData)`
- `jsonDataAsPythonValue` - {'isCat': True, 'miceCaught': 0, 'name': 'Zophie', 'felineIQ': None}

Write JSON with JSON.dumps()

- `pythonValue = {'isCat': True, 'miceCaught': 0, 'name': 'Zophie', 'felineIQ': None}`
- `import json`
- `stringOfJsonData = json.dumps(pythonValue)`
- `stringOfJsonData` - '{"isCat": true, "felineIQ": null, "miceCaught": 0, "name": "Zophie" }'

Ch 15 – Keeping Time, Scheduling Tasks, and Launching Programs

Time objects

Three different types of values used to represent time:

- A Unix epoch timestamp (used by the time module) is a float or integer value of the number of seconds since 12 AM on January 1, 1970, UTC.
- A datetime object (of the datetime module) has integers stored in the attributes year, month, day, hour, minute, and second.
- A timedelta object (of the datetime module) represents a time duration, rather than a specific moment.

Time functions

Time functions and their parameters and return values:

- The time.time() function returns an epoch timestamp float value of the current moment.
- The time.sleep(seconds) function stops the program for the amount of seconds specified by the seconds argument.
- The datetime.datetime(year, month, day, hour, minute, second) function returns a datetime object of the moment specified by the arguments. If hour, minute, or second arguments are not provided, they default to 0.
- The datetime.datetime.now() function returns a datetime object of the current moment.
- The datetime.datetime.fromtimestamp(epoch) function returns a datetime object of the moment represented by the epoch timestamp argument.
- The datetime.timedelta(weeks, days, hours, minutes, seconds, milliseconds, microseconds) function returns a timedelta object representing a duration of time. The function’s keyword arguments are all optional and do not include month or year.
- The total_seconds() method for timedelta objects returns the number of seconds the timedelta object represents.
- The strftime(format) method returns a string of the time represented by the datetime object in a custom format that’s based on the format string.
- The datetime.datetime.strptime(time_string, format) function returns a datetime object of the moment specified by time_string, parsed using the format string argument.

Multi Threading

Make a thread - calling the threading.Thread() function

```
import threading, time
print('Start of program.')

def takeANap():
       time.sleep(5)
       print('Wake up!')

threadObj = threading.Thread(target=takeANap)
threadObj.start()

print('End of program.')
```

Pass arguments and separator into thread

- `threadObj = threading.Thread(target=print, args=['Cats', 'Dogs', 'Frogs'], kwargs={'sep': ' & '})`

Launch other programs from python

- `import subprocess`
- `proc = subprocess.Popen('C:\\Windows\\System32\\calc.exe')`

proc.poll()
proc.wait()

Opening processes with Popen()

can pass command line arguments to processes you create with Popen().
pass a list as the sole argument to Popen). First string in this list will be the executable filename of the program you want to launch. All the subsequent strings will be the command line arguments to pass to the program when it starts

- `subprocess.Popen(['C:\\Windows\\notepad.exe', 'C:\\hello.txt'])` - Open notepad and the hello.txt file when called
- `subprocess.Popen(['C:\\python34\\python.exe', 'hello.py'])` - Open python and the python script when called
- `webbrowser.open()` - Open web browser
- `subprocess.Popen(['open', '/Applications/Calculator.app/'])` - Open a file with its default application (mac os) with ‘open’

Ch 16 – Sending Email and Text Messages

SMTP - sending emails to others

IMAP - retrieving emails sent to you

SMTP

SMTP send email example

- `import smtplib`
- `smtpObj = smtplib.SMTP('smtp.example.com', 587)`
- `smtpObj.ehlo()` - (250, b'mx.example.com at your service, [216.172.148.131]\nSIZE 35882577\n8BITMIME\nSTARTTLS\nENHANCEDSTATUSCODES\nCHUNKING')
- `smtpObj.starttls()` - (220, b'2.0.0 Ready to start TLS')
- `smtpObj.login('bob@example.com', ' MY_SECRET_PASSWORD')` - (235, b'2.7.0 Accepted')
- `smtpObj.sendmail('bob@example.com', 'alice@example.com', 'Subject: So long.\nDear Alice, so long and thanks for all the fish. Sincerely, Bob')` - {}
- `smtpObj.quit()` - (221, b'2.0.0 closing connection ko10sm23097611pbd.52 - gsmtp')

Connect to SMTP Server

Smtp server, then port name. 587 is almost always the port name unless specified

If above step does not work, create an SMTP object using smtplib.SMTP_SSL() and port 465 instead

Common SMTP Servers

Gmail - smtp.gmail.com

Outlook.com/Hotmail.com - smtp-mail.outlook.com

Yahoo Mail - smtp.mail.yahoo.com

AT&T - smpt.mail.att.net (port 465)

Comcast - smtp.comcast.net

Verizon - smtp.verizon.net (port 465)

- `import smtplib`
- `smtpObj = smtplib.SMTP('smtp.gmail.com', 587)`

Sending the SMTP “Hello” Message (ping the server to make sure its active and connected)

- `smtpObj.ehlo()` - (250, b'mx.google.com at your service, [216.172.148.131]\nSIZE 35882577\n8BITMIME\nSTARTTLS\nENHANCEDSTATUSCODES\nCHUNKING')

Starting TLS Encryption

If connecting via SMTP and port 587, start the encryption on SMTP connection. 220 is successful return value

- `smtpObj.starttls()` - (220, b'2.0.0 Ready to start TLS')

Log in to SMTP Server

- `smtpObj.login(' my_email_address@gmail.com ', ' MY_SECRET_PASSWORD ')`

DO NOT, DO NOT HARDCODE PASSWORDS INTO SOURCE CODE. Call input() and have the user type in the password

Send Email

Three arguments

1. Your email address as a string (for the email's "from" address)
2. The recipient's email address as a string or a list of strings for multiple recipients (for the "to" address)
3. The email body as a string - tart of the email body string must begin with 'Subject: \n' for the subject line of the email. The '\n' newline character separates the subject line from the main body of the email.

Return value ifs an empty dict if successful. Will list recipients that have failed if failed

- `smtpObj.sendmail(' my_email_address@gmail.com ', ' recipient@example.com ', 'Subject: So long.\nDear Alice, so long and thanks for all the fish. Sincerely, Bob')` - {}

Disconnect SMTP – 221 if successful

- `smtpObj.quit()`

IMAP

IMAP retrieving email example

- `import imapclient`
- `imapObj = imapclient.IMAPClient('imap.gmail.com', ssl=True)`
- `imapObj.login(' my_email_address@gmail.com ', ' MY_SECRET_PASSWORD ')` - 'my_email_address@gmail.com Jane Doe authenticated (Success)'
- `imapObj.select_folder('INBOX', readonly=True)`
- `UIDs = imapObj.search(['SINCE 05-Jul-2014'])`
- `UIDs` - [40032, 40033, 40034, 40035, 40036, 40037, 40038, 40039, 40040, 40041]
- `rawMessages = imapObj.fetch([40041], ['BODY[]', 'FLAGS'])`

- `import pyzmail`
- `message = pyzmail.PyzMessage.factory(rawMessages[40041]['BODY[]'])`
- `message.get_subject()` - 'Hello!'
- `message.get_addresses('from')` - [('Edward Snowden', 'esnowden@nsa.gov')]
- `message.get_addresses('to')` - [(Jane Doe', 'jdoe@example.com')]
- `message.get_addresses('cc')` - []
- `message.get_addresses('bcc')` - []
- `message.text_part != None` - True
- `message.text_part.get_payload().decode(message.text_part.charset)` - 'Follow the money.\r\n\r\n-Ed\r\n'
- `message.html_part != None` - True
- `message.html_part.get_payload().decode(message.html_part.charset)` - '/
  /
  So long, and thanks for all the fish!/
  /

/
-Al/
\r\n'

- `imapObj.logout()`

Common IMAP Servers

Gmail - imap.gmail.com
Outlook.com/Hotmail.com - imap-mail.outlook.com
Yahoo Mail - imap.mail.yahoo.com
AT&T - imap.mail.att.net
Comcast - imap.comcast.net
Verizon - incoming.verizon.net

Connect to IMAP Server

Email providers need SSL encryption so pass SSL = true

- `import imapclient`
- `imapObj = imapclient.IMAPClient('imap.gmail.com', ssl=True)`

Log in to IMAP server

- `imapObj.login(' my_email_address@gmail.com ', ' MY_SECRET_PASSWORD ')`

DO NOT, DO NOT HARDCODE PASSWORDS INTO SOURCE CODE. Call input() and have the user type in the password

Search for an Email

Select a folder to search through

- `imapObj.select_folder(foldername, readonly=True)`

Call the IMAPClient object's search() method

IMAP search keywords - list of strings, each formatted to the IMAP's search keys

'ALL'
Returns all messages in the folder. You may run in to imaplib size limits if you request all the messages in a large folder. See Size Limits.

- `'BEFORE date', 'ON date', 'SINCE date'`
- `'SUBJECT string', 'BODY string', 'TEXT string'`
- `'FROM string', 'TO string', 'CC string', 'BCC string'`
- `'SEEN', 'UNSEEN'`
- `'ANSWERED', 'UNANSWERED'`
- `'DELETED', 'UNDELETED'`
- `'DRAFT', 'UNDRAFT'` - Returns all messages with and without the \Draft flag, respectively. Draft messages are usually kept in a separate Drafts folder rather than in the INBOX folder.
- `'FLAGGED', 'UNFLAGGED'` - Returns all messages with and without the \Flagged flag, respectively. This flag is usually used to mark email messages as "Important" or "Urgent."
- `'LARGER N', 'SMALLER N'` - Returns all messages larger or smaller than N bytes, respectively.
- `'NOT search-key'`
- `OR search-key1 search-key2'`

Examples

- `imapObj.search(['ALL'])`. Returns every message in the currently selected folder.
- `imapObj.search(['ON 05-Jul-2015'])`. Returns every message sent on July 5, 2015.
- `imapObj.search(['SINCE 01-Jan-2015', 'BEFORE 01-Feb-2015', 'UNSEEN'])`. Returns every message sent in January 2015 that is unread. (Note that this means on and after January 1 and up to but not including February 1.)
- `imapObj.search(['SINCE 01-Jan-2015', 'FROM alice@example.com'])`. Returns every message from alice@example.com sent since the start of 2015.
- `imapObj.search(['SINCE 01-Jan-2015', 'NOT FROM alice@example.com'])`. Returns every message sent from everyone except alice@example.com since the start of 2015.
- `imapObj.search(['OR FROM alice@example.com FROM bob@example.com'])`. Returns every message ever sent from alice@example.com or bob@example.com.
- `imapObj.search(['FROM alice@example.com', 'FROM bob@example.com'])`. Trick example! This search will never return any messages, because messages must match all search keywords. Since there can be only one "from" address, it is impossible for a message to be from both alice@example.com and bob@example.com.

UIDs

- `UIDs = imapObj.search(['SINCE 05-Jul-2015'])`
- `UIDs` - [40032, 40033, 40034, 40035, 40036, 40037, 40038, 40039, 40040, 40041]
  Return value is the emails themselves but rather unique IDs (UIDs) for the emails, as integer values. You can then pass these UIDs to the fetch() method to obtain the email content

Fetching an Email and Marking It As Read

Fetch from list of UIDs and download all the body content for the emails specified in your UID list

```
rawMessages = imapObj.fetch(UIDs, ['BODY[]'])
import pprint
pprint.pprint(rawMessages)
{40040: {'BODY[]': 'Delivered-To: my_email_address@gmail.com\r\n'
                   'Received: by 10.76.71.167 with SMTP id '
--snip--
                   '\r\n'
                   '------=_Part_6000970_707736290.1404819487066--\r\n',
         'SEQ': 5430}}
```

If you do want emails to be marked as read when you fetch them, you will need to pass readonly=False to select_folder()

`imapObj.select_folder('INBOX', readonly=False)`

Getting Email Addresses from a Raw Message

raw messages returned from the fetch() method need to be processed

pyzmail module parses these raw messages and returns them as PyzMessage objects, which make the subject, body, "To" field, "From" field, etc. accessible

Body - Emails can be sent as plaintext, HTML, or both.

Plaintext emails contain only text, while HTML emails can have colors, fonts, images, and other features that make the email message look like a small web page.

- `import pyzmail`
- `message = pyzmail.PyzMessage.factory(rawMessages[40041]['BODY[]'])`
- `message.get_subject()` - 'Hello!'

- `message.get_addresses('from')` - [('Edward Snowden', 'esnowden@nsa.gov')]
- `message.get_addresses('to')` - [(Jane Doe', 'my_email_address@gmail.com')]
- `message.get_addresses('cc')` - []
- `message.get_addresses('bcc')` - []
- `message.text_part.get_payload().decode(message.text_part.charset)` - 'So long, and thanks for all the fish!\r\n\r\n-Al\r\n'
- `message.html_part.get_payload().decode(message.html_part.charset)-'`
  So long, and thanks for all the fish!

  -Al
  \r\n'

Delete Emails

- `imapObj.select_folder('INBOX', readonly=False)`
- `UIDs = imapObj.search(['ON 09-Jul-2015'])`
- `UIDs` - [40066]
- `imapObj.delete_messages(UIDs)` - {40066: ('\Seen', '\Deleted')}
- `imapObj.expunge()` - calling expunge() then permanently deletes messages with the \Deleted flag. ('Success', [(5452, 'EXISTS')])

Disconnecting from the IMAP Server

- `imapObj.logout()`

Sending Text Messages with Twilio

Signing Up for a Twilio Account

Sending Text Messages

- `from twilio.rest import TwilioRestClient`
- `accountSID` = 'ACxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
- `authToken` = 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
- `twilioCli` = TwilioRestClient(accountSID, authToken)
- `myTwilioNumber` = '+14955551234'
- `myCellPhone` = '+14955558888'
- `message` = twilioCli.messages.create(body='Mr. Watson - Come here - I want to see you.', from_=myTwilioNumber, to=myCellPhone)
- `message.to` - '+14955558888'
- `message.from` - '+14955551234'
- `message.body` - 'Mr. Watson - Come here - I want to see you.'
- `message.status` - 'queued'
- `message.date_created` - datetime.datetime(2015, 7, 8, 1, 36, 18)
- `message.date_sent == None` - True
- `message.sid` - 'SM09520de7639ba3af137c6fcb7c5f4b51'
- `updatedMessage = twilioCli.messages.get(message.sid)`
- `updatedMessage.status` - 'delivered'
- `updatedMessage.date_sent` - datetime.datetime(2015, 7, 8, 1, 36, 18)

Ch 17 – Manipulating Images

Image color codes

- `from PIL import ImageColor`
- `ImageColor.getcolor('red', 'RGBA')` - (255, 0, 0, 255)

RBGA color codes 4 numbers
Box tuple values: (left, top, right, bottom) pixels. Origin on top left

Pillow to manipulate images

- `from PIL import Image`
- `catIm = Image.open('zophie.png')`

Actions on Image Data Type

- `catIm.size` - (816, 1088)
- `width, height = catIm.size`
- `width` - 816
- `height` - 1088
- `catIm.filename` - 'zophie.png'
- `catIm.format` - 'PNG'
- `catIm.format_description` - 'Portable network graphics'
- `catIm.save('zophie.jpg')`

Create a new image with one background color

- `from PIL import Image`
- `im = Image.new('RGBA', (100, 200), 'purple')`
- `im.save('purpleImage.png')`
- `im2 = Image.new('RGBA', (20, 20))` - no color specified so transparent
- `im2.save('transparentImage.png')`

Cropping Images

- `croppedIm = catIm.crop((335, 345, 565, 560))`
- `croppedIm.save('cropped.png')`

Copying and Pasting Images onto Other Images

- `catIm = Image.open('zophie.png')`
- `catCopyIm = catIm.copy()`
- `faceIm = catIm.crop((335, 345, 565, 560))`
- `faceIm.size` - (230, 215)
- `catCopyIm.paste(faceIm, (0, 0))`
- `catCopyIm.paste(faceIm, (400, 500))`
- `catCopyIm.save('pasted.png')`

Resize an image

- `width, height = catIm.size`
- `quartersizedIm = catIm.resize((int(width / 2), int(height / 2)))`
- `quartersizedIm.save('quartersized.png')`
- `svelteIm = catIm.resize((width, height + 300))`
- `svelteIm.save('svelte.png')`

Rotating and Flipping Images (rotate number of degrees)

- `catIm.rotate(90).save('rotated90.png')`
- `catIm.rotate(180).save('rotated180.png')`
- `catIm.rotate(270).save('rotated270.png')`
- `catIm.transpose(Image.FLIP_LEFT_RIGHT).save('horizontal_flip.png')`
- `catIm.transpose(Image.FLIP_TOP_BOTTOM).save('vertical_flip.png')`

Changing Individual Pixels

Use getpixel() and putpixel() to get and set pixels

- `im = Image.new('RGBA', (100, 100))`
- `im.getpixel((0, 0))` - (0, 0, 0, 0)
-

```
for x in range(100):
           for y in range(50):
             im.putpixel((x, y), (210, 210, 210))
```

- `from PIL import ImageColor`
-

```
for x in range(100):
           for y in range(50, 100):
             im.putpixel((x, y), ImageColor.getcolor('darkgray', 'RGBA'))
```

- `im.getpixel((0, 0))` - (210, 210, 210, 255)
- `im.getpixel((0, 50))` - (169, 169, 169, 255)
- `im.save('putPixel.png')`

Drawing on Images

Drawing Shapes

- `from PIL import Image, ImageDraw`
- `im = Image.new('RGBA', (200, 200), 'white')`
- `draw = ImageDraw.Draw(im)`
- `draw.line([(0, 0), (199, 0), (199, 199), (0, 199), (0, 0)], fill='black')`
- `draw.rectangle((20, 30, 60, 60), fill='blue')`
- `draw.ellipse((120, 30, 160, 60), fill='red')`
- `draw.polygon(((57, 87), (79, 62), (94, 85), (120, 90), (103, 113)), fill='brown')`

- `for i in range(100, 200, 10):`
  `draw.line([(i, 0), (200, i - 100)], fill='green')`
- `im.save('drawing.png')`

Drawing Text

- `from PIL import Image, ImageDraw, ImageFont`
- `import os`
- `im = Image.new('RGBA', (200, 200), 'white')`
- `draw = ImageDraw.Draw(im)`
- `draw.text((20, 150), 'Hello', fill='purple')`
- `fontsFolder = 'FONT_FOLDER' # e.g. 'Library/Fonts'`
- `arialFont = ImageFont.truetype(os.path.join(fontsFolder, 'arial.ttf'), 32)`
- `draw.text((100, 150), 'Howdy', fill='gray', font=arialFont)`
- `im.save('text.png')`

Ch 18 – Controlling the Keyboard and Mouse with GUI Automation

- `import pyautogui`

Kill gui automation program - SHIFT-OPTION-Q

- `pyautogui.PAUSE = 1.5` - Adding a wait
- `pyautogui.FAILSAFE = True` - Adding a failsafe to prevent moving offscreen
- `width, height = pyautogui.size()` - Store width and height of screen
- `pyautogui.position()` - Getting mouse position

PyAutoGui Functions

- `pyautogui.moveTo(x, y)` - Moves the mouse cursor to the given x and y coordinates.
- `pyautogui.moveRel(xOffset, yOffset)` - Moves the mouse cursor relative to its current position.
- `pyautogui.dragTo(x, y)` - Moves the mouse cursor while the left button is held down.
- `pyautogui.dragRel(xOffset, yOffset)` - Moves the mouse cursor relative to its current position while the left button is held down.
- `pyautogui.click(x, y, button)` - Simulates a click (left button by default).
- `pyautogui.rightClick()` - Simulates a right-button click.
- `pyautogui.middleClick()` - Simulates a middle-button click.
- `pyautogui.doubleClick()` - Simulates a double left-button click.
- `pyautogui.mouseDown(x, y, button)` - Simulates pressing down the given button at the position x, y.
- `pyautogui.mouseUp(x, y, button)` - Simulates releasing the given button at the position x, y.
- `pyautogui.scroll(units)` - Simulates the scroll wheel. A positive argument scrolls up; a negative argument scrolls down.
- `pyautogui.typewrite(message)` - Types the characters in the given message string.
- `pyautogui.typewrite([key1, key2, key3])` - Types the given keyboard key strings.
- `pyautogui.press(key)` - Presses the given keyboard key string.
- `pyautogui.keyDown(key)` - Simulates pressing down the given keyboard key.
- `pyautogui.keyUp(key)` - Simulates releasing the given keyboard key.
- `pyautogui.hotkey([key1, key2, key3])`- Simulates pressing the given keyboard key strings down in order and then releasing them in reverse order.
  pyautogui.screenshot(). Returns a screenshot as an Image object. (See Chapter 17 for information on Image objects.)

Code names for keys

-'a', 'b', 'c', 'A', 'B', 'C', '1', '2', '3', '!', '@', '#', and so on

-'enter' (or 'return' or '\n')

-'esc'

-'shiftleft', 'shiftright'

-'altleft', 'altright'

-'ctrlleft', 'ctrlright'

-'tab' (or '\t')

-'backspace', 'delete'

-'pageup', 'pagedown'

-'home', 'end'

-'up', 'down', 'left', 'right'

-'f1', 'f2', 'f3', and so on

-'volumemute', 'volumedown', 'volumeup'

-'pause'

-'capslock', 'numlock', 'scrolllock'

-'insert'

-'printscreen'

-'winleft', 'winright'

-'command'