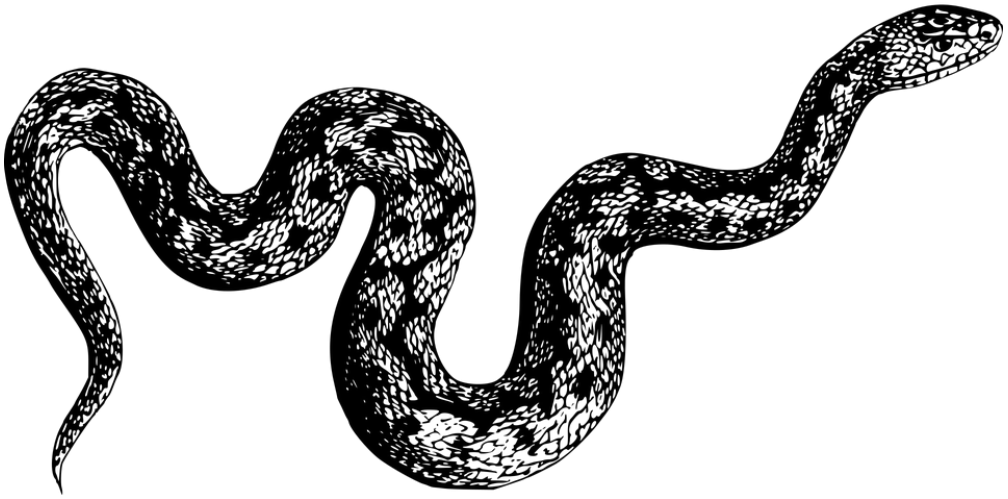
Anatomy of a Python Script

The first few times that I tried to learn Python, it felt like learning a bunch of made-up rules about an imaginary universe. It turns out that Python is kind of like an imaginary universe with made-up rules. That's part of what makes Python and programming languages so much fun.

But it can also make learning Python difficult if you don't really know what the imaginary universe looks like, or how it functions, or how it relates to your universe and your specific goals — such as doing text analysis or making a Twitter bot or creating a network visualization.

In this lesson, we're going to demonstrate what Python looks like in action, so you can get a feel for its structure and flow. Don't get too bogged down in the details for now. Just try to get a sense — at an abstract level — of how Python works and how you might use it.



Below is a chunk of Python code. These lines, when put together, do something simple yet important. They count and display the most frequent words in a text file. The example below specifically counts and displays the 40 most frequent words in Charlotte Perkins Gilman's short story "The Yellow Wallpaper" (1892).

Contents

```
import re
from collections import Counter

def split_into_words(any_chunk_of_text):
    lowercase_text = any_chunk_of_text.lower()
    split_words = re.split("\W+", lowercase_text)
    return split_words

filepath_of_text = "../texts/literature/The-Yellow-Wallpaper_Charlotte-Perkins-
Gilman.txt"
number_of_desired_words = 40

stopwords = ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you',
'your', 'yours',
 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', 'her', 'hers',
 'herself', 'it', 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves',
 'what', 'which', 'who', 'whom', 'this', 'that', 'these', 'those', 'am', 'is', 'are',
 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does',
 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until',
 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into',
 'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down',
 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once',
'here',
 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more',
 'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so',
 'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', 'should', 'now', 've',
'll', 'amp']

full_text = open(filepath_of_text, encoding="utf-8").read()

all_the_words = split_into_words(full_text)
meaningful_words = [word for word in all_the_words if word not in stopwords]
meaningful_words_tally = Counter(meaningful_words)
most_frequent_meaningful_words =
meaningful_words_tally.most_common(number_of_desired_words)

most_frequent_meaningful_words
```

```
[('john', 45),
 ('one', 33),
 ('said', 30),
 ('would', 27),
 ('get', 24),
 ('see', 24),
 ('room', 24),
 ('pattern', 24),
 ('paper', 23),
 ('like', 21),
 ('little', 20),
 ('much', 16),
 ('good', 16),
 ('think', 16),
 ('well', 15),
 ('know', 15),
 ('go', 15),
 ('really', 14),
 ('thing', 14),
 ('wallpaper', 13),
```

Calculating word frequency is a very basic form of computational text analysis. Typically, it's not terribly interesting on its own, especially with a single short text. But calculating word frequency *is* important, and it's at the center of most text analysis approaches, even far more complicated ones.

> **Python Review**
>
> It's important to emphasize that the code above is just *one* way to count words in a text file with Python. This is not the one right way. There is no right way to count words in a text file or to do anything else in Python.
>
> Rather than asking "Is this code *right*?", you want to ask yourself:
>
> - "Is this code efficient?"
> - "Is this code readable?"
> - "Does this code help me accomplish my goal?"
>
> Sometimes you'll prioritize one of these concerns over another. Maybe your code isn't as efficient as humanly possible, but if it gets the job done, and you understand it, then you might not care about maximum efficiency. Our main goal for this class is to study and make arguments about culture, not (necessarily) to become the most efficient software developers.

The Anatomy of a Python Script

Import Libraries/Packages/Modules

Ready for some great Python news? You don't have to code everything by yourself from scratch! Many other people have written Python code that you can `import` into your own code, which will save you time and do a lot of work behind-the-scenes.

```
import re
from collections import Counter
```

We call the code written and packaged up by other people a "library," "package," or "module." We'll talk more about them in a later lesson. For now simply know that you `import` libraries/packages/modules at the very top of a Python script for later use.

- `Counter` will help me count words
- `re`, short for regular expressions, is basically a fancy find-and-replace that will help me split "The Yellow Wallpaper" into individual words and get rid of trailing punctuation

Define Functions

After `import`ing modules and libraries, you typically `def`ine your "functions." Functions are a nifty way to bundle up code so that you can use them again later. Functions also keep your code neat and tidy.

```
def split_into_words(any_chunk_of_text):
    words = re.split("\W+", any_chunk_of_text.lower())
    return words
```

Here we're making a function called `split_into_words`, which takes in any chunk of text, transforms that text to lower-case, and splits the text into a list of clean words without punctuation or spaces. We're not actually using the function yet.

Define Filepaths and Assign Variables

Here we establish some variables that we're going to use later.

```
filepath_of_text = "../texts/literature/The-Yellow-Wallpaper_Charlotte-Perkins-Gilman.txt"
number_of_desired_words = 40
stopwords = ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', 'your', 'yours',
 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', 'her', 'hers',
 'herself', 'it', 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves',
 'what', 'which', 'who', 'whom', 'this', 'that', 'these', 'those', 'am', 'is', 'are',
 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does',
 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until',
 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into',
 'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down',
 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once', 'here',
 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more',
 'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so',
 'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', 'should', 'now', 've', 'll', 'amp']
```

We'll need the filepath of the short story in order to read it, so we make a variable called `filepath_of_text`. We also make a variable called `number_of_desired_words`, which will eventually tell the script how many words to display.

Lastly we also make a variable called `stopwords` and plug in a list of common English language "stop words"—that is, a list of some of the most frequently occurring English language words. Stop words are typically removed from a text before computational analysis in order to shift the focus to less frequently occurring, more "meaningful" words.

Food for Thought

What are the consequences of excluding stopwords from our analysis? When might we want to include such words in an analysis?

Read in File

```
full_text = open(filepath_of_text, encoding="utf-8").read()
```

The line above opens Charlotte Perkins Gilman's "The Yellow Wallpaper," reads in the novel, and assigns it to the variable `full_text`.

Manipulate and Analyze File

To count the words in "The Yellow Wallpaper," we need to break the full text into individual words. Below we call the function `split_into_words`, which we created earlier, and use it to split the `full_text` of the story into individual words. Then we assign this value to the variable `all_the_words`.

```
all_the_words = split_into_words(full_text)
```

Then we remove stopwords from our list. The line of code below makes a new list of words that includes every word in `all_the_words` if it does *not* appear in `stopwords` (aka it nixes the stopwords).

```
meaningful_words = [word for word in all_the_words if word not in stopwords]
```

Now we're ready to count! We plug `meaningful_words` into our `Counter`, which gives us a tally of how many times each word in the story appears.

```
meaningful_words_tally = Counter(meaningful_words)
```

Output Results

Lastly, we pull out the top 40 most frequently occurring words from our complete tally. We make one final variable and grab our top `number_of_desired_words`, which we previously established as `40`.

```
most_frequent_meaningful_words =
meaningful_words_tally.most_common(number_of_desired_words)
```

We can display our results by running a cell with the variable name `most_frequent_meaningful_words`.

```
most_frequent_meaningful_words
```

```
[('john', 45),
 ('one', 33),
 ('said', 30),
 ('would', 27),
 ('get', 24),
 ('see', 24),
 ('room', 24),
 ('pattern', 24),
 ('paper', 23),
 ('like', 21),
 ('little', 20),
 ('much', 16),
 ('good', 16),
 ('think', 16),
 ('well', 15),
 ('know', 15),
 ('go', 15),
 ('really', 14),
 ('thing', 14),
 ('wallpaper', 13),
```

Or we can display our results by `print`ing the variable `most_frequent_meaningful_words`.

```
print(most_frequent_meaningful_words)
```

```
[('john', 45), ('one', 33), ('said', 30), ('would', 27), ('get', 24), ('see', 24),
('room', 24), ('pattern', 24), ('paper', 23), ('like', 21), ('little', 20), ('much',
16), ('good', 16), ('think', 16), ('well', 15), ('know', 15), ('go', 15), ('really',
14), ('thing', 14), ('wallpaper', 13), ('night', 13), ('long', 12), ('course', 12),
('things', 12), ('take', 12), ('always', 12), ('could', 12), ('jennie', 12),
('great', 11), ('says', 11), ('feel', 11), ('even', 11), ('used', 11), ('dear', 11),
('time', 11), ('enough', 11), ('away', 11), ('want', 11), ('never', 10), ('must',
10)]
```

Comments

Lines that begin with a hash symbol `#` are ignored from the execution of the code. You can thus use a hash symbol `#` to insert human language comments directly into the code — notes or instructions to yourself and others.

In some cases, you might want to write a long comment. To insert a multi-line comment, you can insert the comment between three quotations marks `"""` `"""`.

```
"""
Example Python code for
calculating word frequency
in a text file
"""

# Import Libraries and Modules


import re
from collections import Counter

# Define Functions

def split_into_words(any_chunk_of_text):
    lowercase_text = any_chunk_of_text.lower()
    split_words = re.split("\W+", lowercase_text)
    return split_words

# Define Filepaths and Assign Variables

filepath_of_text = "../texts/literature/The-Yellow-Wallpaper_Charlotte-Perkins-
Gilman.txt"
number_of_desired_words = 40

stopwords = ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you',
'your', 'yours',
 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', 'her', 'hers',
 'herself', 'it', 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves',
 'what', 'which', 'who', 'whom', 'this', 'that', 'these', 'those', 'am', 'is', 'are',
 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does',
 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until',
 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into',
 'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down',
 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once',
'here',
 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more',
 'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so',
 'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', 'should', 'now', 've',
'll', 'amp']

# Read in File

full_text = open(filepath_of_text, encoding="utf-8").read()

# Manipulate and Analyze File

all_the_words = split_into_words(full_text)
meaningful_words = [word for word in all_the_words if word not in stopwords]
meaningful_words_tally = Counter(meaningful_words)
most_frequent_meaningful_words =
meaningful_words_tally.most_common(number_of_desired_words)

# Output Results

most_frequent_meaningful_words
```

```
[('john', 45),
 ('one', 33),
 ('said', 30),
 ('would', 27),
 ('get', 24),
 ('see', 24),
 ('room', 24),
 ('pattern', 24),
 ('paper', 23),
 ('like', 21),
 ('little', 20),
 ('much', 16),
 ('good', 16),
 ('think', 16),
 ('well', 15),
 ('know', 15),
 ('go', 15),
 ('really', 14),
 ('thing', 14),
 ('wallpaper', 13),
 ('night', 13)
```

The Life of a Python Script

Jupyter Notebook / JupyterLab

The primary way that we're going to write and run Python in this class is through JupyterLab and Jupyter notebooks. As we've already covered, Jupyter notebooks are documents that can combine live code, explanatory text, and nice displays of data, which makes them great for teaching and learning.

But it's also a fully functional way to run Python. By running a cell of Python code in a Jupyter notebook, you can:

- read files from your computer and write files to your computer
- make and save a bar chart
- collect data from YouTube or Spotify
- programmatically tweet from a Twitter bot account
- and a lot more!

For example, by adding two lines of code at the bottom of our script, I can output the most frequent words from "The Yellow Wallpaper" into a text file.

```
def split_into_words(any_chunk_of_text):
    lowercase_text = any_chunk_of_text.lower()
    split_words = re.split("\W+", lowercase_text)
    return split_words

filepath_of_text = "../texts/literature/The-Yellow-Wallpaper.txt"
nltk_stop_words = stopwords.words("english")
number_of_desired_words = 40

full_text = open(filepath_of_text, encoding="utf-8").read()

all_the_words = split_into_words(full_text)
meaningful_words = [word for word in all_the_words if word not in nltk_stop_words]
meaningful_words_tally = Counter(meaningful_words)
most_frequent_meaningful_words = meaningful_words_tally.most_common(number_of_desired_words)

with open("most-frequent-words-Yellow-Wallpaper.txt", "w") as file_object:
    file_object.write(str(most_frequent_meaningful_words))
```

Text Editor —> Command Line

You can also run a Python script by writing it in a text editor and then running it from the command line.

If you copy and paste the code above into a simple text editor (like TextEdit or NotePad) and name the file with the extension ".py" (the file extension for Python code), you should be able to run the script from your command line.

word_frequency_Yellow_Wallpaper.py

```
#Import Libraries and Modules

import re
from collections import Counter
from nltk.corpus import stopwords


# Define Functions

def split_into_words(any_chunk_of_text):
    lowercase_text = any_chunk_of_text.lower()
    split_words = re.split("\W+", lowercase_text)
    return split_words

# Define Filepaths and Assign Variables

filepath_of_text = "../texts/literature/The-Yellow-Wallpaper.txt"
nltk_stop_words = stopwords.words("english")
number_of_desired_words = 40

# Read in File

full_text = open(filepath_of_text).read()

# Manipulate and Analyze File

all_the_words = split_into_words(full_text)
meaningful_words = [word for word in all_the_words if word not in
nltk_stop_words]
meaningful_words_tally = Counter(meaningful_words)
most_frequent_meaningful_words =
meaningful_words_tally.most_common(number_of_desired_words)
```

All you need to do is call python with the name of the Python file (and make sure that the script includes the correct file path).

```
!python word_frequency_Yellow_Wallpaper.py
```

```
[('john', 45), ('one', 33), ('said', 30), ('would', 27), ('get', 24), ('see', 24),
('room', 24), ('pattern', 24), ('paper', 23), ('like', 21), ('little', 20), ('much',
16), ('good', 16), ('think', 16), ('well', 15), ('know', 15), ('go', 15), ('really',
14), ('thing', 14), ('wallpaper', 13), ('night', 13), ('long', 12), ('course', 12),
('things', 12), ('take', 12), ('always', 12), ('could', 12), ('jennie', 12),
('great', 11), ('says', 11), ('feel', 11), ('even', 11), ('used', 11), ('dear', 11),
('time', 11), ('enough', 11), ('away', 11), ('want', 11), ('never', 10), ('must',
10)]
```

Though it's possible to write Python from TextEdit, it's not very common, because it's a pain. It's much more common to write Python code in a text editor like Atom, as shown below. You can see that there's all sorts of formatting and functionality that makes the code writing faster and easier.

word_frequency_Yellow_Wallpaper.py — ~/Intro-Cultural-Analytics

word_frequency_Yellow_Wallpaper.py

```
#Import Libraries and Modules

import re
from collections import Counter
from nltk.corpus import stopwords


# Define Functions

def split_into_words(any_chunk_of_text):
    lowercase_text = any_chunk_of_text.lower()
    split_words = re.split("\W+", lowercase_text)
    return split_words

# Define Filepaths and Assign Variables

filepath_of_text = "../texts/literature/The-Yellow-Wallpaper.txt"
nltk_stop_words = stopwords.words("english")
number_of_desired_words = 40

# Read in File

full_text = open(filepath_of_text).read()

# Manipulate and Analyze File
```

You can also write Python scripts such that they can work with different files or any file you want it to. With a few small alterations, our word frequency script can crunch numbers for Grimms Fairy Tales

```
!python word_frequency.py ../texts/literature/Grimms-Fairy-Tales.txt
```

```
[('said', 911), ('little', 498), ('one', 454), ('king', 438), ('went', 385), ('came',
359), ('go', 266), ('away', 239), ('old', 233), ('man', 225), ('good', 211), ('took',
210), ('two', 209), ('woman', 199), ('saw', 193), ('could', 193), ('come', 186),
('time', 184), ('day', 180), ('would', 178), ('well', 177), ('_', 163), ('home',
163), ('back', 161), ('shall', 159), ('eyes', 157), ('three', 153), ('daughter',
150), ('mother', 145), ('house', 142), ('thought', 139), ('must', 139), ('forest',
138), ('great', 136), ('cried', 134), ('take', 134), ('long', 133), ('door', 132),
('nothing', 130), ('let', 130)]
```

or Louisa May Alcott's *Little Women*

```
!python word_frequency.py ../texts/literature/Little-Women_Louisa-May-Alcott.txt
```

```
[('jo', 1395), ('one', 898), ('said', 839), ('little', 758), ('meg', 697), ('amy',
659), ('laurie', 608), ('like', 604), ('beth', 482), ('good', 479), ('would', 440),
('see', 422), ('m', 407), ('go', 400), ('old', 394), ('mother', 382), ('never', 375),
('much', 375), ('well', 371), ('could', 359), ('away', 340), ('time', 330), ('mr',
324), ('know', 321), ('march', 320), ('made', 303), ('home', 286), ('young', 285),
('girls', 281), ('think', 280), ('day', 279), ('come', 278), ('say', 277), ('went',
274), ('came', 273), ('dear', 265), ('got', 260), ('face', 260), ('make', 257),
('asked', 255)]
```

or any other text your heart desires!

By Melanie Walsh