

Intro to Creative Computing

Week 8: Files

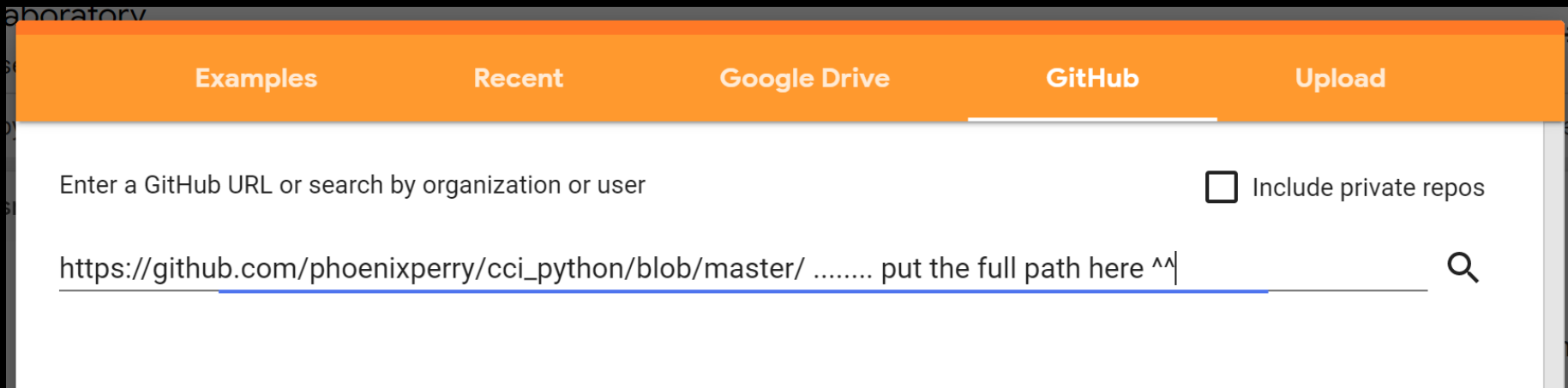
Your projects!

Today's class

- Intro to files
- *Pause*
- Exercises with files
- *Pause*
- Work on the projects

Google Colab

- We can test examples together, go to <http://colab.research.google.com/> and fill the path to the *.ipynb* file:

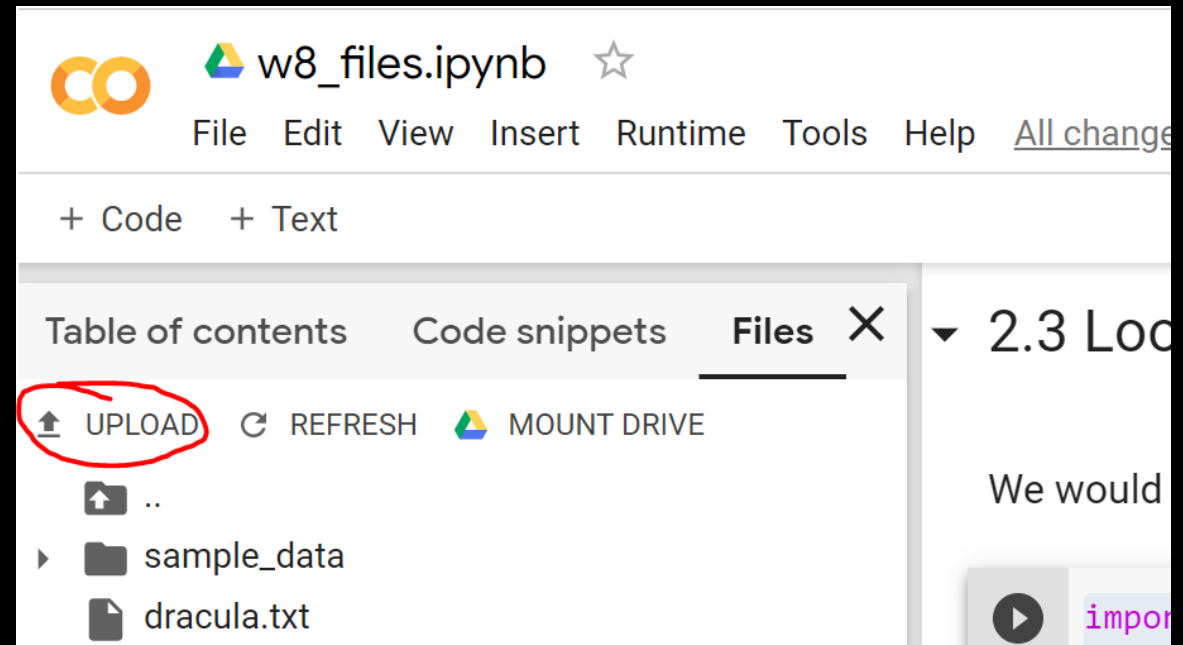
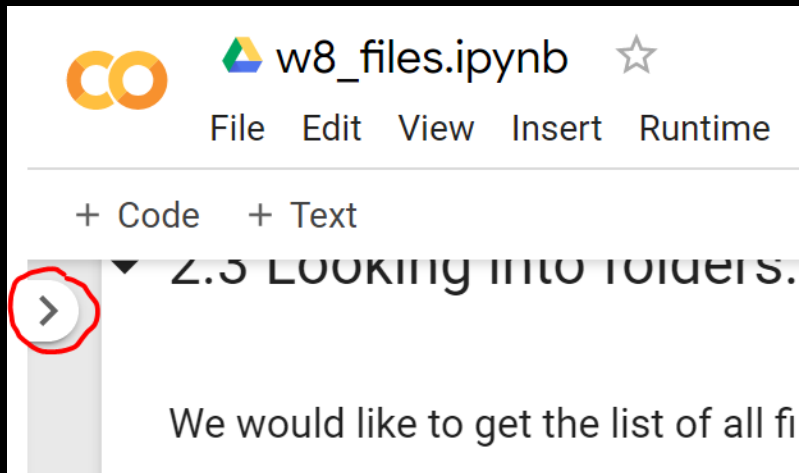


The screenshot shows the 'GitHub' tab in the Google Colab interface. At the top, there is a navigation bar with five tabs: 'Examples', 'Recent', 'Google Drive', 'GitHub' (which is selected), and 'Upload'. Below the tabs, there is a text input field with the placeholder text 'Enter a GitHub URL or search by organization or user'. To the right of this field is a checkbox labeled 'Include private repos'. Below the input field, there is a text input field containing the URL 'https://github.com/phoenixperry/cci_python/blob/master/' followed by a blue underline and the text '..... put the full path here ^^'. To the right of this field is a magnifying glass icon.

- Path:
https://github.com/phoenixperry/cci_python/blob/master/week08/class_code/w8_files/main_examples.ipynb

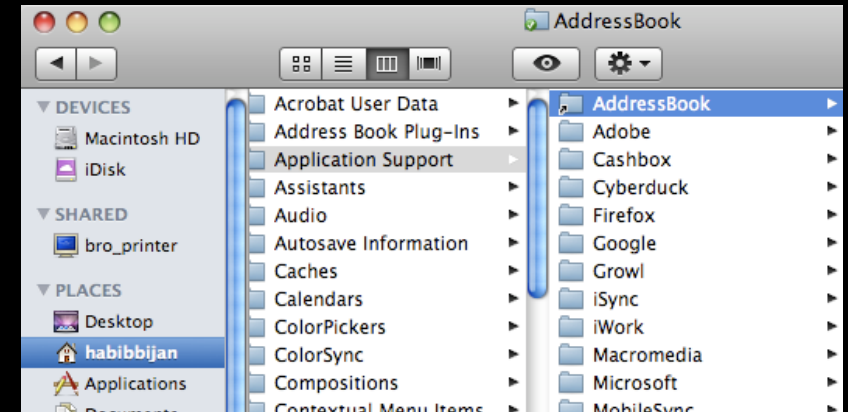
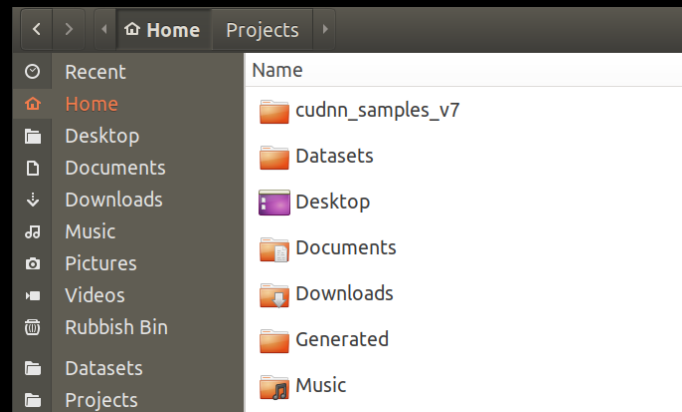
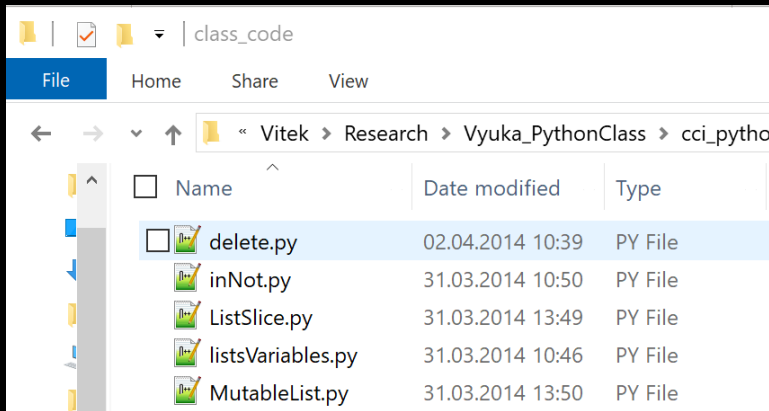
Files in Google Colab

- There are multiple ways to upload your files to Colab and also to check them, the **easiest** one is through their **Files dialog**:



Files

- Files are what we usually work with:



- Path to a file:
 - Relative: "delete.py"
 - Absolute: "D:/Vitek/Research/Vyuka_PythonClass/cci_python/week07/class_code/delete.py"
- Relative* goes from where our python file is run from
- Absolute* is the full operation system path

Loading text files

- Loading *.txt* files – line by line or in one go

```
infile = open("dracula.txt", "r")  
  
for line in infile:  
    print(line)  
  
infile.close()
```

```
infile = open("dracula.txt", "r")  
  
one_long_string = infile.read()  
  
infile.close()  
  
print(one_long_string)
```

- Test these on Colab ^ ... Can you load your own text files?

Saving files

- Saving a string to a file:

```
outfile = open("saved_file.txt", "w")  
  
outfile.write(message)  
  
outfile.close()
```

- Test this on Colab

Append to file

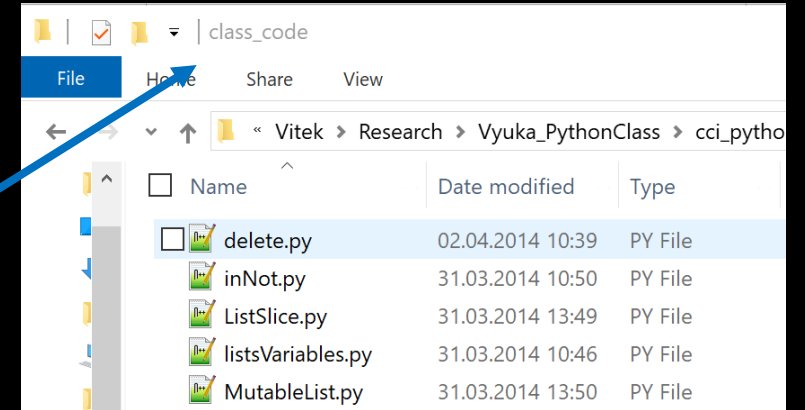
- Appending means adding to already existing file – or making a new file

```
# this adds at the end of existing file!  
outfile = open("saved_file.txt", "a")  
outfile.write("Additional notes")  
outfile.close()
```

```
infile = open("dracula.txt", "r")           # read  
outfile1 = open("saved_file.txt", "w")      # write  
outfile2 = open("saved_file.txt", "a")      # append
```

Folders

- Files inside a folder



```
import os
files = os.listdir("../class_code")
print( files )
```

- Use path which works on your operating system

Folders

- Make a new folder

```
import os  
os.mkdir("results")
```

Reading

- Go through the **chapter 11** on files at:

<https://runestone.academy/runestone/books/published/thinkcspy/Files/toctree.html>

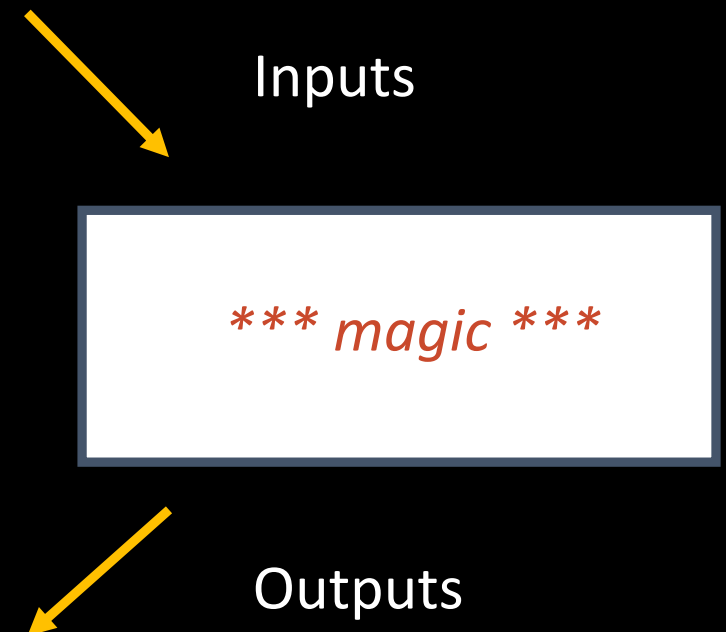
Pause 1

Functions *revisited*

- Why are functions cool?

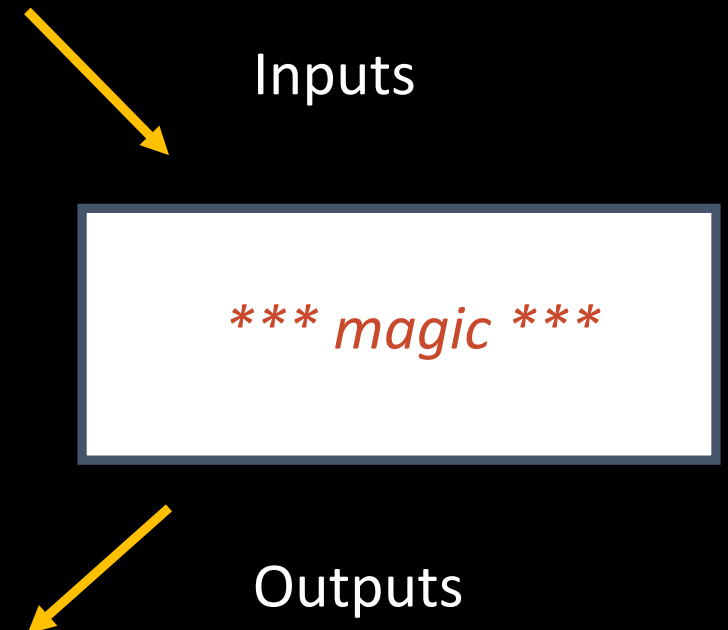
Functions *revisited*

- Function – has input, does something and then gives something back (sort of like a *coffee machine* you give it water and coffee beans and it gives you hot beverage)



Functions *revisited*

- Function – has input, does something and then gives something back (sort of like a *coffee machine* you give it water and coffee beans and it gives you hot beverage)

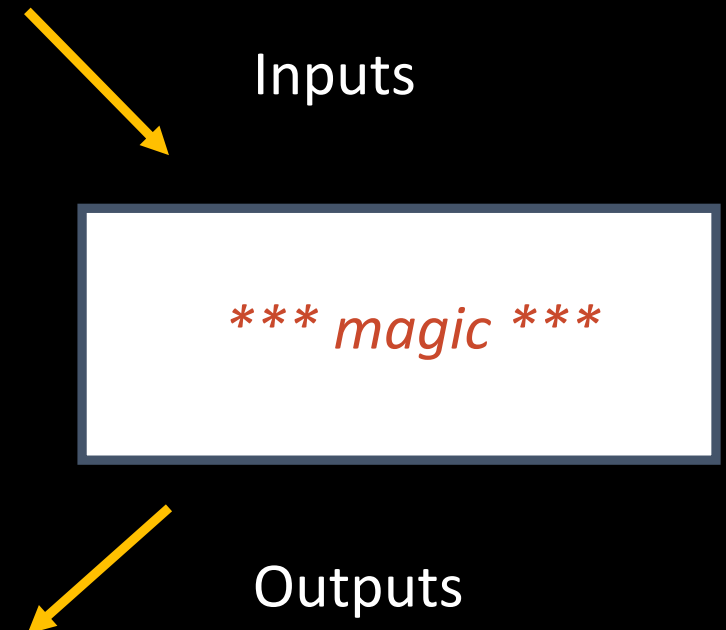


Functions *revisited*

- Function – has input, does something and then gives something back (sort of like a *coffee machine* you give it water and coffee beans and it gives you hot beverage)

```
def my_function(some_input):  
    # do something with it  
    # ...  
    return some_output
```

```
def coffee_machine(input_water, input_beans):  
    # do something with it  
    # ...  
    return coffee_beverage
```



Functions *revisited*

- Can be called repeatedly (reused block of code):

```
def text_analysis(text_input):  
    # something magical with it  
    # ...  
    return analysis_output
```

```
analysis_alice = text_analysis("Alice's Adventures in Wonderland ...")  
print(analysis_alice)
```

```
analysis_frank = text_analysis("Frankenstein; Or, The Modern Prometheus ...")  
print(analysis_frank)
```

Exercise

- Task 1: Process the text of Alice in Wonderland (download it from Gutenberg here: <https://www.gutenberg.org/files/11/11.txt>), count number of times these words appear:

- clock, mushroom, alice, rabbit, queen, lost

Save the result as a string into "alice1.txt" file.

~~PS: Don't use text.count("word")~~ << ok, use it

Hint: When saving a number convert it to string with `str(10)` => "10"

Example:

```
count_apples = 0
# ... get the counts

message = "Text contains "+str(count_apples)+" apples."
# Then save this into file using the example from these slides
# (slide: Saving files)
```

Exercise

- Task 2: Using the same file, count the occurrences of these words across the whole text (paragraph by paragraph) and make a list of these occurrences.

Hint 1: To split the text into paragraphs use:

```
paragraphs = text.split("\n\n")
```

Hint 2: Keep a list of numbers and add to it:

```
numbers = []  
numbers.append( 13 )
```

Example list with counts:

```
[1, 0, 2, 0, 0, 0, 4, 1, 0, 0, 0]
```

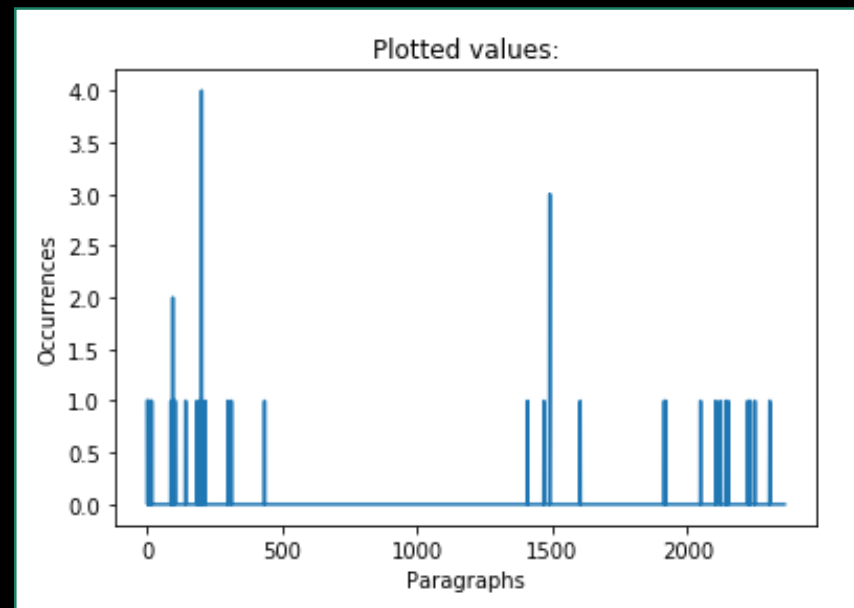
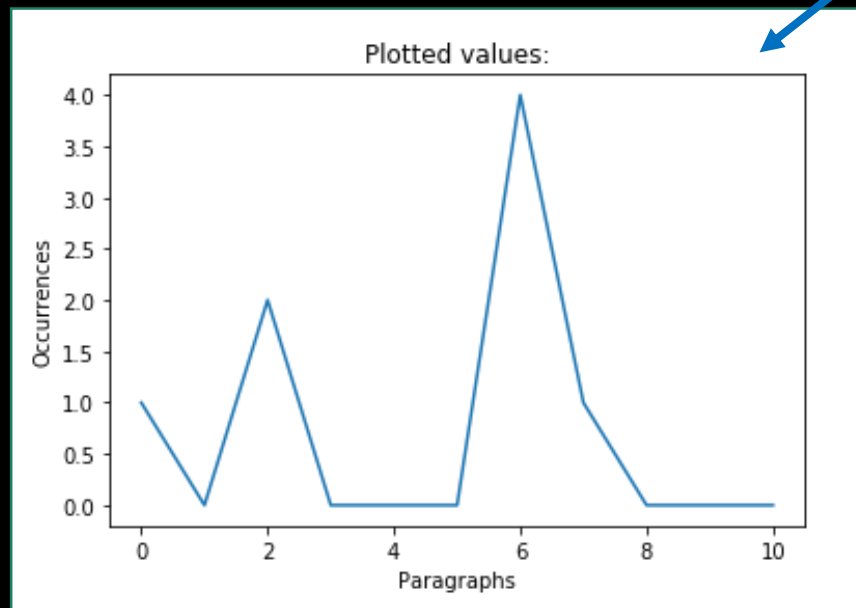
Bonus: Use my function in w8_plot_list.py to show these as a graph!

Bonus: Use my function in w8_plot_list.py to show these as a graph!

```
list = [1, 0, 2, 0, 0, 0, 4, 1, 0, 0, 0]

from w8_plot_list.py import ViteksPlotter

# see the magicks happen with:
ViteksPlotter(list)
```



Exercise

- Task 3 (Advanced): We often get data sources with a lot of small text files (imagine for example having tweets saved as individual .txt files). Get the names of all files in a folder (example in these slides), select only text files (with “.txt” in the name) and load them all.

Hint: You can start with downloading few texts from Gutenberg.

- Then get which is the most common word across all of these files.
(PS: this is the *advanced* advanced part :D)

Pause 2

Text analysis suggestions (1)

- Occurrence of a word in a long text:

```
long_text = "Alice's Adventures in Wonderland ..."  
occurrence_alice = long_text.count("Alice")  
print(occurrence_alice)
```


Text analysis suggestions (2)

- Going paragraph by paragraph and counting occurrences:

```
long_text = "Alice's Adventures in Wonderland ..."  
  
paragraphs = long_text.split("\n\n")  
occurences_list = []  
  
for paragraph in paragraphs:  
    count = paragraph.count("Alice")  
    occurences_list.append(count)  
  
print(occurences_list)
```

Text analysis suggestions (3)

- Comparing with wordlists

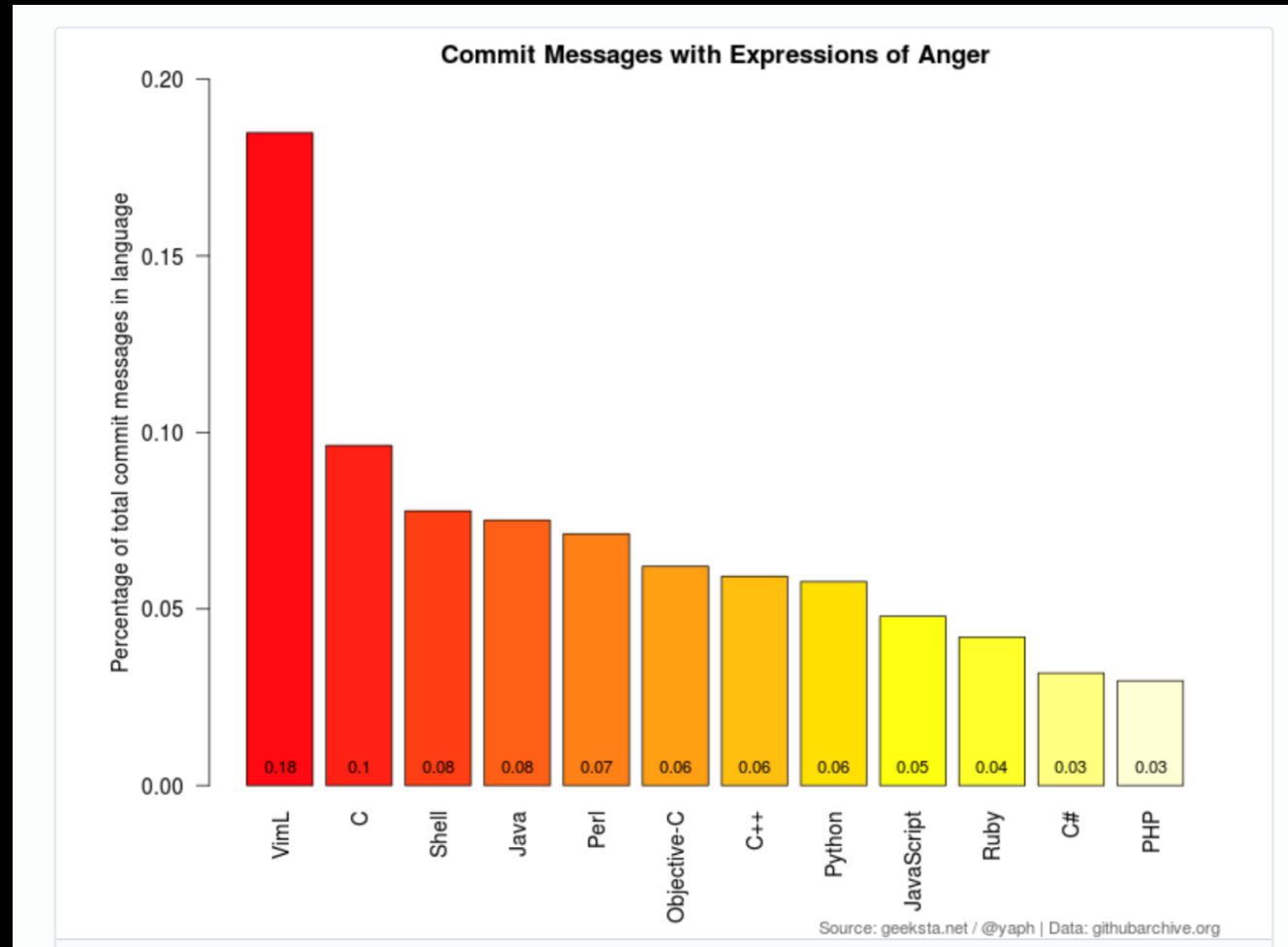
```
tweet_text = "@Holly_Looya @divermam Ok. Ben--Congratulations on your great interview  
with @IngrahamAngle discussing the disgusting story she made up facts and a total joke  
compared to @HuffingtonPost. http://somelink.com/"

wordlist = ["made up", "wall", "wow", "joke"]
count = 0
for check_word in wordlist:
    if check_word in tweet_text:
        count = count + 1

print("Text contains", count, "flagged words!")
```

- Please excuse this simplified example. The sample tweet text comes from a generative "Automatic Trump" model at <https://filiph.github.io/markov/>

Text analysis suggestions (3)



- Better example with wordlists?

Next?

- Cool thing: Plotting in Python using Matplotlib (so how did it work?)
- Plug everything you learned today into your project!

Homework

- Do the exercise from <https://runestone.academy/runestone/books/published/thinkcspy/Files/Exercises.html>
 - Exercises 1, 2, 3 and 5 look fun!
 - PS: the solutions are there as well, but try writing them yourself first!