Finding Text in Python

27 January 2022
Assignment 1 out now; due Wednesday at 10 p.m.
Unless otherwise indicated, assignments (not worksheets!) can be completed in pairs.
Introduction to Python development
#!/usr/bin/env python3

print("It's your friend, Emacs")
#!/usr/bin/env python3

print("It's your friend, Emacs")

; chmod a+x emacs.py
; ./emacs.py

It's your friend, Emacs
;
M-x shell

M means the “meta” key, which is what you and I call “Escape”.

```
#!/usr/bin/env python3
print("It's your friend, Emacs")
```
Other options:

vim
gedit
VSCode
...

Jupyter notebooks (like Google Colab) are great, and they’re widely used in NLP and data science. Feel free to use them for experimentation or when you’re working on your final project, but the assignments you turn in should be normal, well-written Python files (.py), not notebook (.ipynb).
Regular expressions in Python
RE functions to know:

`re.search`
`re.match`
`re.finditer`
`re.findall`
`re.compile`
`re.sub`
import re

s = "Hello there"

m = re.search(r"\b(t?here)\b", s)

print(m.group(1))
import re

s = "Hello there"

m = re.search(r"\b(t?here)\b", s)

print(m.group(1))
import re

s = "Hello there"

m = re.match(r"\b(t?here)\b", s)

print(m.group(1))
import re

s = "Hello there"

m = re.match(r"\b(t?here)\b", s)

print(m.group(1))

Error! re.match only matches from the beginning of the string. It’s equivalent to starting the RE with ^.
import re

s = "Hello there, hello here, hello everywhere"

for m in re.finditer(r"\b(t?here)\b", s):
    print(m.group(1))
import re

s = "Hello there, hello here, hello everywhere"

for match in re.findall(r"\b(there)\b", s):
    print(match)
import re

s = "Hello there"

prog = re.compile(r"(Hello|howdy)"

m = prog.match("Hello there")
print(m.group(1))

m = prog.match("Howdy partner")
print(m.group(1))
import re

s = "Hello there"

prog = re.compile(r"(Hello|howdy)"

m = prog.match("Hello there")
print(m.group(1))

m = prog.match("Howdy partner")
print(m.group(1))

Compiling lets us efficiently re-use an RE.
import re

s = "Hello there"

t = re.sub("(Hello|Hi) there", r"\1", s)

print(t)
Practice: Information Extraction
Output will be triples (entity1, relation, entity2), e.g.,

("Grace Hopper", "born in", "1906")
#!/usr/bin/env python3

import fileinput
import re

prog = re.compile(r"((?:(?:(?:[A-Z][a-z]+ )+)(born .*)\([0-9]{4}\))\))")

for line in fileinput.input():
    m = prog.search(line)
    name = m.group(1).strip()
    year = m.group(2)
    print(f"("{name}", "born in", "{year}"\)")
Debugging in Python
For some assignments this semester, code may take a long time to run – minutes or even hours.

This can make debugging with `print` statements painful:

Since it might be several minutes before your code runs into an error, if it takes a few tries to find which `print` statements will find the information you need, you could spend lots of time waiting in between each try.
The `ipdb` (Interactive Python DeBugger) module provides a nice interface to Python's built-in debugger, `pdb`. 
Whenever we have a Python module we want to install, we can use the `pip` command:

```
$ pip3 install ipdb
```
And then to use ipdb, we first import it:

```python
import ipdb
```
What **ipdb** allows us to do is inline debugging in our code – that is, to run code such that, when it reaches a point you’re interested in or it hits an error, it will pause right there and give you an IPython console you can use to interact with your variables in context using Python.
If you have a particular point in your code where you want to be able to interact with it for debugging, you can add

    ipdb.set_trace()

right before where you want to look.
If you’re not quite sure where an error might arrive but would like the chance to inspect the code if an exception is raised, you can use a special “context manager”, which wraps the code you want to execute with an instruction to use the debugger if an exception occurs:

```python
from ipdb import launch_ipdb_on_exception

with launch_ipdb_on_exception():
    ⟨code that might have issues⟩
```
When you run your code the normal way, e.g.,

```
$ ./slackbot.py
```

or

```
$ python3 slackbot.py
```

if there is an exception raised, now it will start the ipdb console.
Once the debugging console starts up, you can use it like you’d use a Python interpreter:

You can print variables from the current step of code execution to see their values,
Define new variables, and even
Write short functions.
The debugging console also has special commands you can use to navigate the code from which the debugger was invoked, including

? \textit{function}

Print the docstring for the specified function

\textbf{next} or \textbf{n}

Run the current line of code and proceed to the next line of that function or the function that called it.

\textbf{step} or \textbf{s}

This is like next, but it runs the current line of code until either a new function is called (at which point it’ll place you in that function’s context) or the next line of the function is reached.

\textbf{continue} or \textbf{c}

Keep running until the next breakpoint or the code completes.

\textbf{p} \textit{variable} or \textbf{pp} \textit{variable}

Prints (or pretty-prints) the variable or code snippet provided.
Assignment 1
Part 1: RegEx Golf
Regex Golf

Warmup - Type a regex in the box.

Match all of these... and none of these...

- sfoot
- catfoot
- dogfoot
- fanfoot
- foody
- foolery
- foolish
- fooster
- footage
- footnot
- footlet
- footage
- footpad
- footway
- hotfoot
- jawfoot
- mfoot
- mfoot
- padfoot
- presfoot
- sfoot
- unfool

- Atlas
- Aymoro
- Iberic
- Mahr
- Gormad
- Sillpan
- altered
- chandoo
- cremel
- crooked
- fardo
- folkay
- forest
- hebemio
- idgah
- manlike
- marly
- palazzl
- sixfold
- terrock
- unfold
“…fixing two kinds of errors: *false positives*, strings that we incorrectly matched … and *false negatives*, strings that we incorrectly missed… Reducing the overall error rate for an application thus involves two antagonistic efforts:

• Increasing *precision* (minimizing false positives)

• Increasing *recall* (minimizing false negatives)’’

Jurafsky & Martin, § 2.1.3
Part 2: Chatbot
See Assignment 1 `slackbot.py`.
Acknowledgments

This class incorporates material from:

Xanda Schofield, Harvey Mudd College