An Introduction to Python for Text Analysis

Marco Sammon

Kellogg School of Management

1/12/2017

Outline

1. Running Python programs in PyCharm
2. Python data types and syntax
3. Text analysis in Python
4. Automated downloading of SEC filings
5. Extracting counts of text characteristics
6. Discussion of Financial Health Economics by Koijen et. al. (2016)

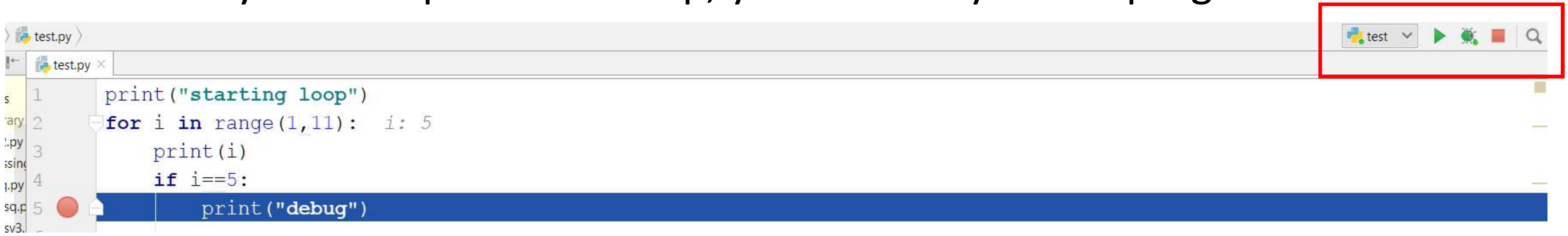
Running Python Programs in PyCharm

Sampleprogram1.py

Sampleprogram2.py

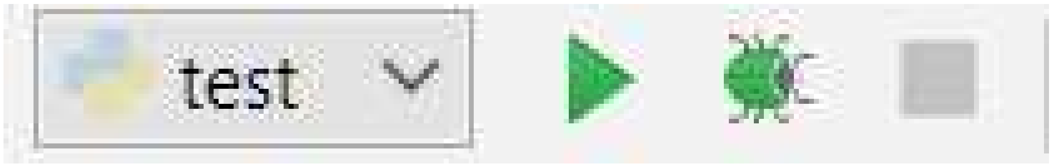
Running Programs

- Once your interpreter is set up, you are ready to run programs



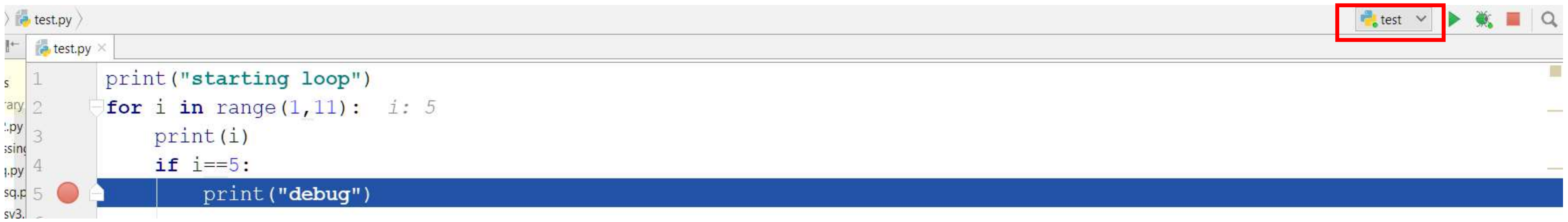
```
print("starting loop")
for i in range(1,11):   i: 5
    print(i)
    if i==5:
        print("debug")
```

Select program   Run   Debug   Stop



Switching Programs

- In PyCharm, switching tabs in the editor does not switch the active program
- There are two ways to switch the active program
    1. Right click in the editor on the program of interest, and select Run
    2. Use the box in the top right of the PyCharm window to select a different program



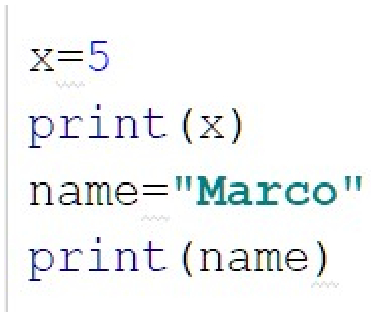
```
print("starting loop")
for i in range(1,11):   i: 5
    print(i)
    if i==5:
        print("debug")
```

Notes on Python 3.X Data Types and Syntax

ifelse.py
Tryexceptcode.py
Datastrctures.py
Callfunction.py

Python Variables and Lists

- Variables
  - Put quotes around strings
    - Both “ and ‘ will work



```
x=5
print(x)
name="Marco"
print(name)
```

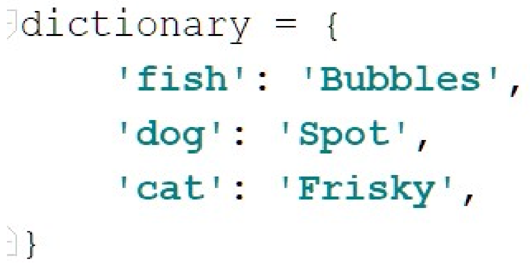
- Lists
  - Example: mylist=[1,1,2,3,5,…]
  - First element of the list, mylist[0]
  - Last element of the list, mylist[-1]

Python Dictionaries

- Dictionaries
  - Has elements and attributes
  - This dictionary has 3 elements
    - Fish, dog and cat
  - To extract an attribute, use:

    dictionary[element]

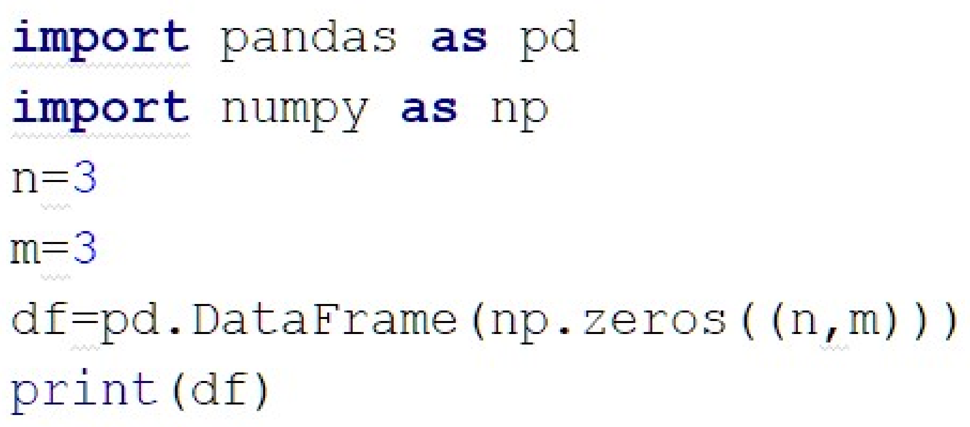
    Example: *dictionary[dog]* will return “Spot”



```
dictionary = {
     'fish': 'Bubbles',
     'dog': 'Spot',
     'cat': 'Frisky',
}
```

Pandas Dataframes

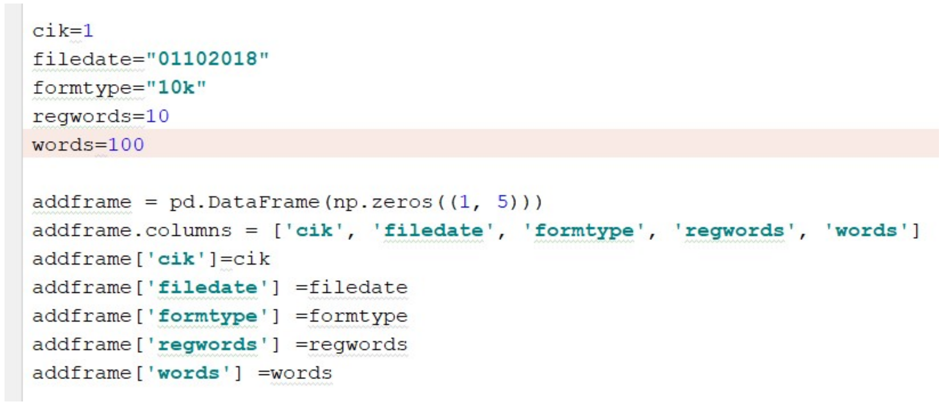
- Use pandas to create a *Data Frame*, which is a matrix, but it can store more than just numbers
  - The usual call for pandas is *import pandas as pd*
  - To create a new data frame, use *pd.DataFrame*
    - To fill it with zeros using *np.zeros*, which creates an n by m matrix of zeros



```
import pandas as pd
import numpy as np
n=3
m=3
df=pd.DataFrame(np.zeros((n,m)))
print(df)
```

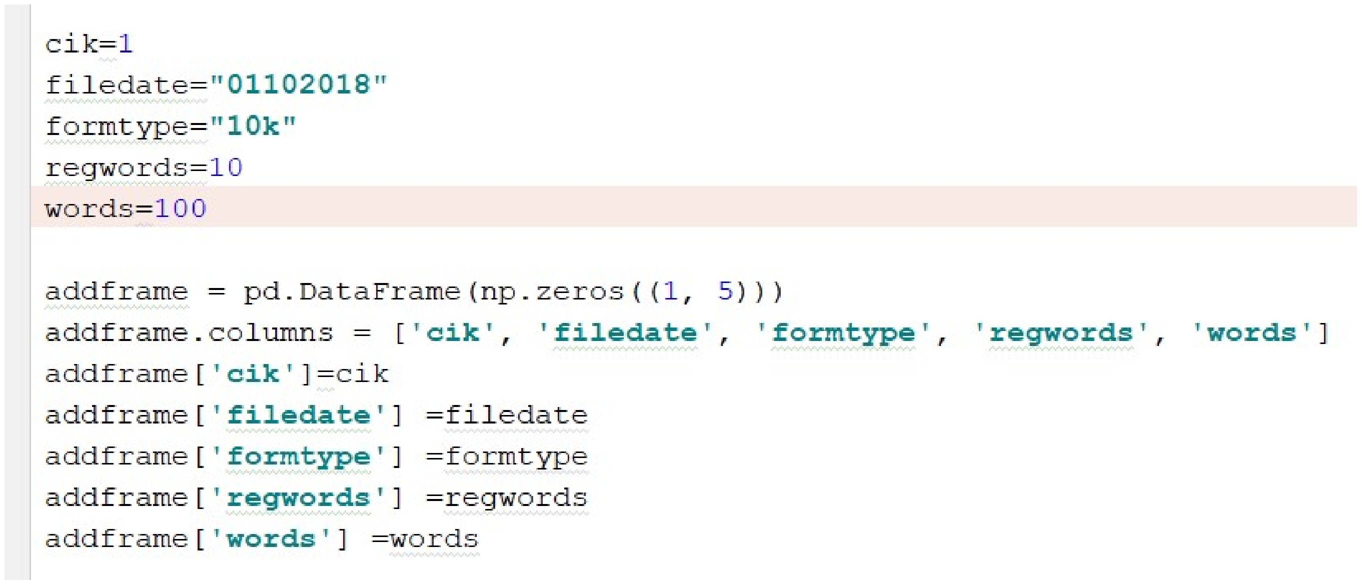
Accessing elements of a data frame with one row

- I like to label the columns, and use the syntax:
    - *Your_data_frame['your_column']*
- To label columns , use *your_data_frame.columns*
- If you try to assign data to a column that doesn't already exist, pandas will create a new column



```
cik=1
filedate="01102018"
formtype="10k"
regwords=10
words=100

addframe = pd.DataFrame(np.zeros((1, 5)))
addframe.columns = ['cik', 'filedate', 'formtype', 'regwords', 'words']
addframe['cik']=cik
addframe['filedate'] =filedate
addframe['formtype'] =formtype
addframe['regwords'] =regwords
addframe['words'] =words
```

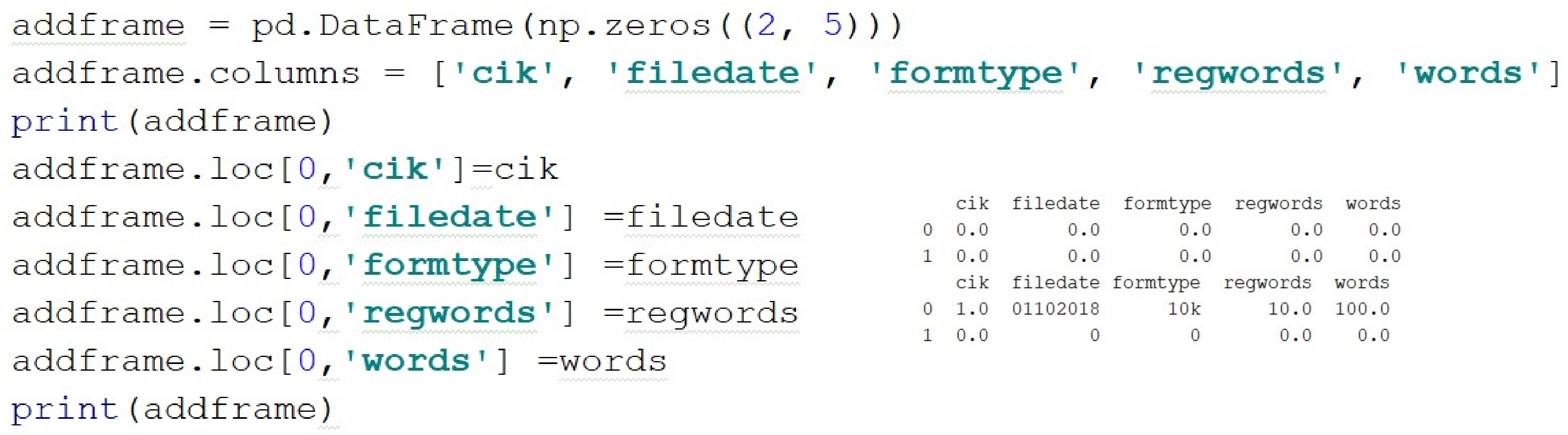
Accessing elements of a data frame with one row



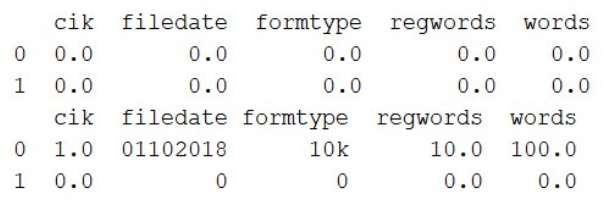
```
cik=1
filedate="01102018"
formtype="10k"
regwords=10
words=100

addframe = pd.DataFrame(np.zeros((1, 5)))
addframe.columns = ['cik', 'filedate', 'formtype', 'regwords', 'words']
addframe['cik']=cik
addframe['filedate'] =filedate
addframe['formtype'] =formtype
addframe['regwords'] =regwords
addframe['words'] =words
```

Data Frames with Multiple Rows

- Similar to working with lists:



```
addframe = pd.DataFrame(np.zeros((2, 5)))
addframe.columns = ['cik', 'filedate', 'formtype', 'regwords', 'words']
print(addframe)
addframe.loc[0,'cik']=cik
addframe.loc[0,'filedate'] =filedate
addframe.loc[0,'formtype'] =formtype
addframe.loc[0,'regwords'] =regwords
addframe.loc[0,'words'] =words
print(addframe)
```

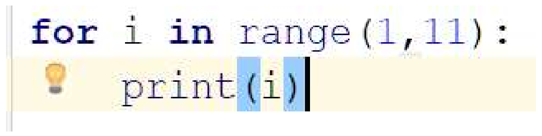


```
   cik  filedate  formtype  regwords  words
0  0.0       0.0       0.0       0.0    0.0
1  0.0       0.0       0.0       0.0    0.0
   cik  filedate formtype  regwords  words
0  1.0  01102018      10k      10.0  100.0
1  0.0         0        0       0.0    0.0
```

For more details on slicing data frames:
https://pandas.pydata.org/pandas-docs/stable/indexing.html

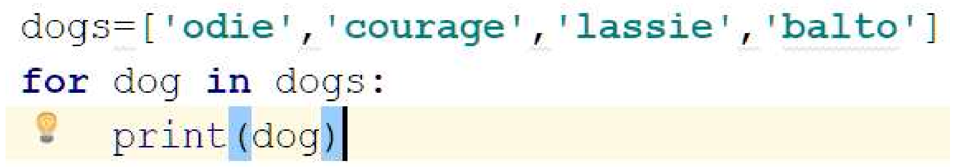
Python Syntax: Loops

- Loop to print integers between 1 and 10
  - Print statements are different than Python 2 – remember to put the variable in parenthesis
  - 1st number in “range” is included, last is not
  - **Don’t forget the colon, and don’t forget to indent**



```
for i in range(1,11):
    print(i)
```

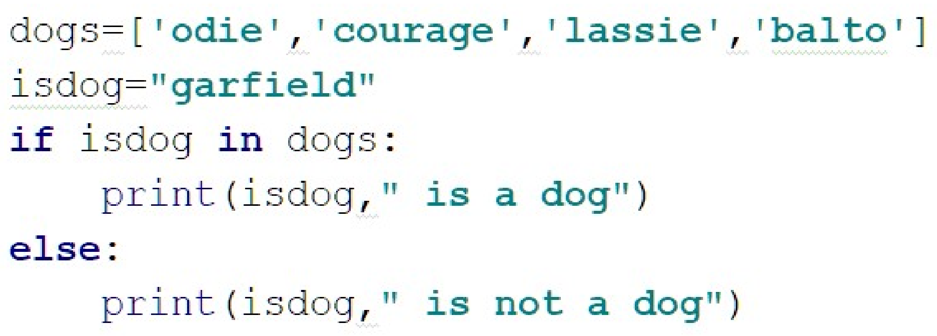
- Loop over items in list:



```
dogs=['odie','courage','lassie','balto']
for dog in dogs:
    print(dog)
```

Python Syntax: If/Else

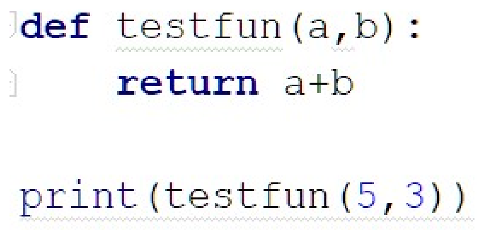
- Check if an item appears in a list, then perform an action
- Again, **don’t forget colons or to indent**



```
dogs=['odie','courage','lassie','balto']
isdog="garfield"
if isdog in dogs:
    print(isdog," is a dog")
else:
    print(isdog," is not a dog")
```

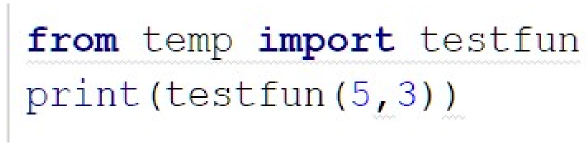
Python Syntax: Functions

- You can write functions and call them in the same file:
  - A function must be defined before it is called



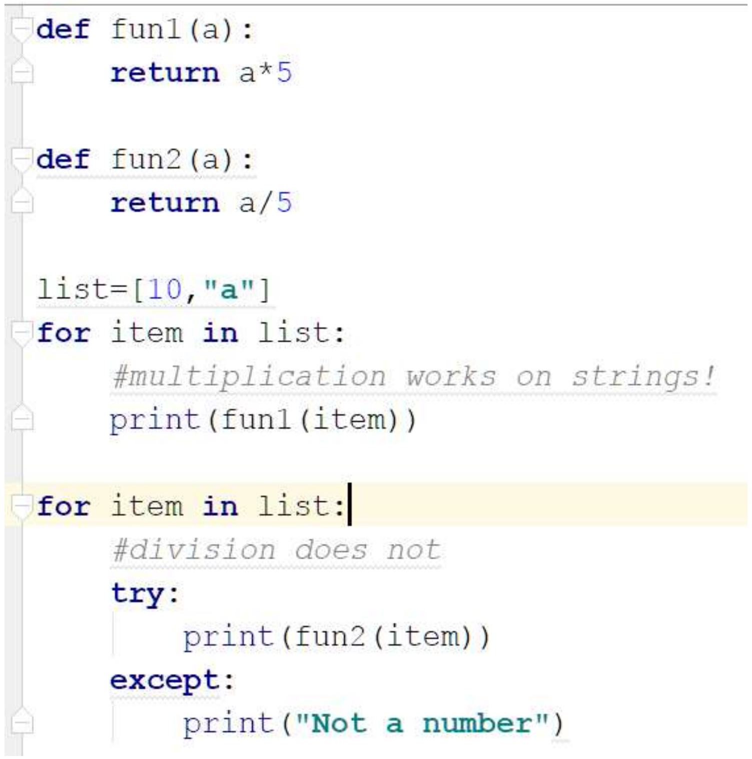
```
def testfun(a,b):
    return a+b

print(testfun(5,3))
```

- The function can also be written and called from another file:
  - *from your_file_name import your_function*



```
from temp import testfun
print(testfun(5,3))
```

Python Syntax: Try/Except



```python
def fun1(a):
    return a*5

def fun2(a):
    return a/5

list=[10,"a"]
for item in list:
    #multiplication works on strings!
    print(fun1(item))

for item in list:
    #division does not
    try:
        print(fun2(item))
    except:
        print("Not a number")
```

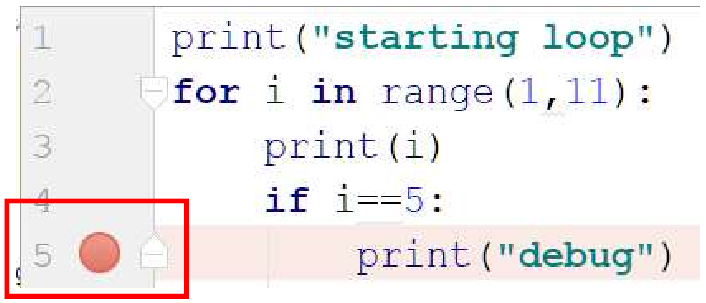
```
50
aaaaa
2.0
Not a number
```

Detour: Debugging

Datastruct2.py

Debugging Programs

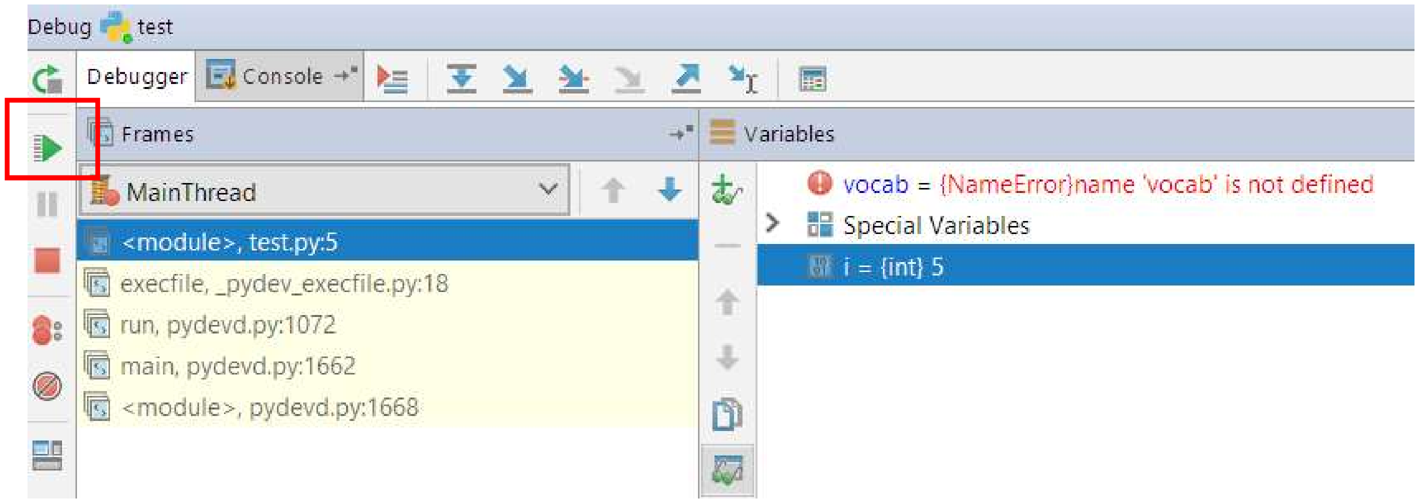
Insert a breakpoint – code will stop here
The line with breakpoint will not run



```
print("starting loop")
for i in range(1,11):
    print(i)
    if i==5:
        print("debug")
```

Debugging Window

You can resume your code with this button

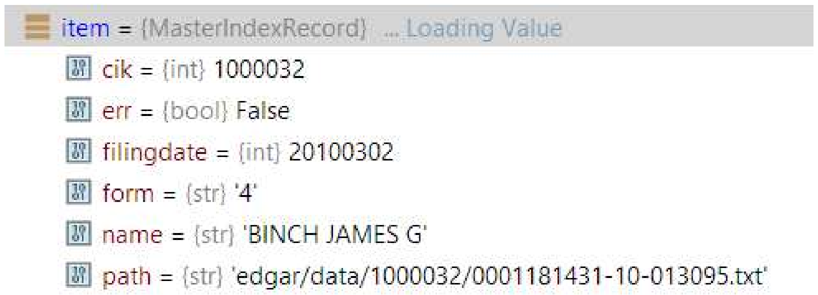


```
<module>, test.py:5
execfile, _pydev_execfile.py:18
run, pydevd.py:1072
main, pydevd.py:1662
<module>, pydevd.py:1668
```

```
vocab = {NameError}name 'vocab' is not defined
Special Variables
i = {int} 5
```

Why Use Debug vs. Run?

- Suppose you get the following error:
  - *AttributeError: ‘YOUR_DICTIONARY' object has no attribute ‘YOUR_DATA’*
- You can use the debugger to find any item’s attributes:



```
item = {MasterIndexRecord} ...Loading Value
    cik = {int} 1000032
    err = {bool} False
    filingdate = {int} 20100302
    form = {str} '4'
    name = {str} 'BINCH JAMES G'
    path = {str} 'edgar/data/1000032/0001181431-10-013095.txt'
```

Text Analysis in Python

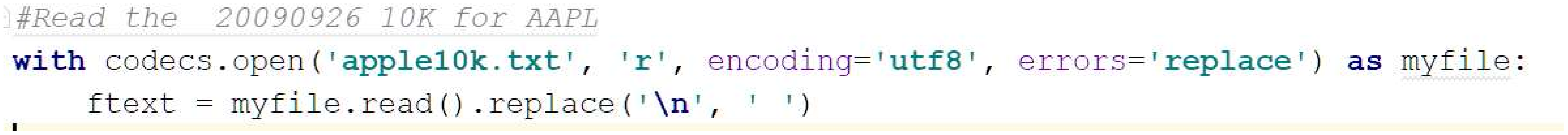
Testriskfactors.py
Regex.py
Wordstems.py
Sectionextract.py

Topics to Cover

- Opening/reading text files
- Writing/saving text files
- Regular Expressions

Opening Files

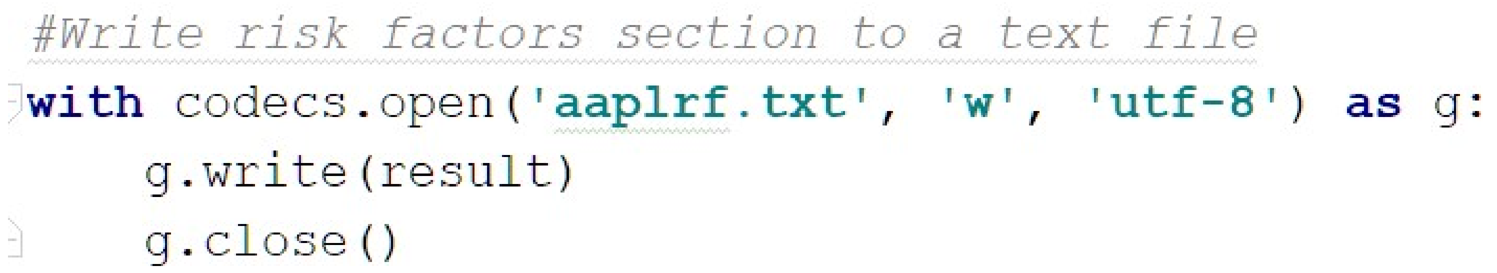
- You can open files with *open* but I use the *codecs* package
  - remember to import codecs
- Arguments for codecs.open:
  - Document name
  - “r” is for reading
  - UTF-8, this is the default encoding for HTML
    - Some characters cannot be represented in UTF-8, the replace command will put a flag character in place of the missing one. ‘ignore’ will it dropped entirely
  - Replacing ‘\n’ removes linebreaks



```
#Read the  20090926 10K for AAPL
with codecs.open('apple10k.txt', 'r', encoding='utf8', errors='replace') as myfile:
    ftext = myfile.read().replace('\n', ' ')
```

Writing Files

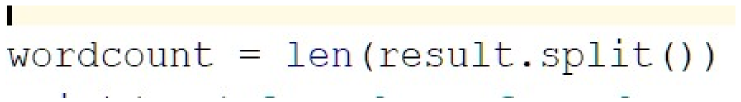
- When saving text to a file, you also use *codecs.open*
  - New document name
  - The “r” has been replaced with a “w” for writing
  - Encoding



```
#Write risk factors section to a text file
with codecs.open('aaplrf.txt', 'w', 'utf-8') as g:
    g.write(result)
    g.close()
```

Regular Expressions

- Counting the number of words in a string
  - *yourstring.split()* breaks a string by spaces, and puts the pieces into a list
  - Len(your list) returns the number of items in a list

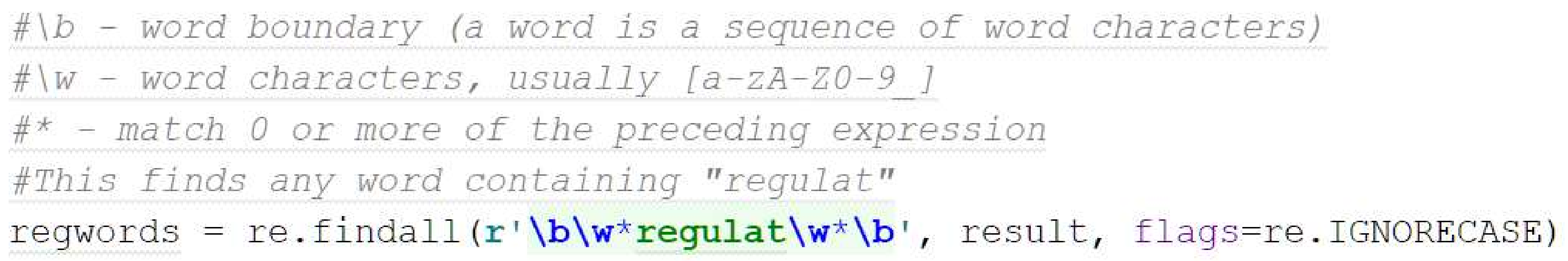


```
wordcount = len(result.split())
```

  - You can pass any argument you want between the parenthesis in *split*. This can be used to split by specific words, paragraphs, etc.
    - Syntax: *yourstring.split(yourdelimiter)*

Regular Expressions: Example

- Finding instances of a particular word with re.findall
  - See https://docs.python.org/3.6/library/re.html for details on regular expressions syntax
  - Setting up the regular expression:

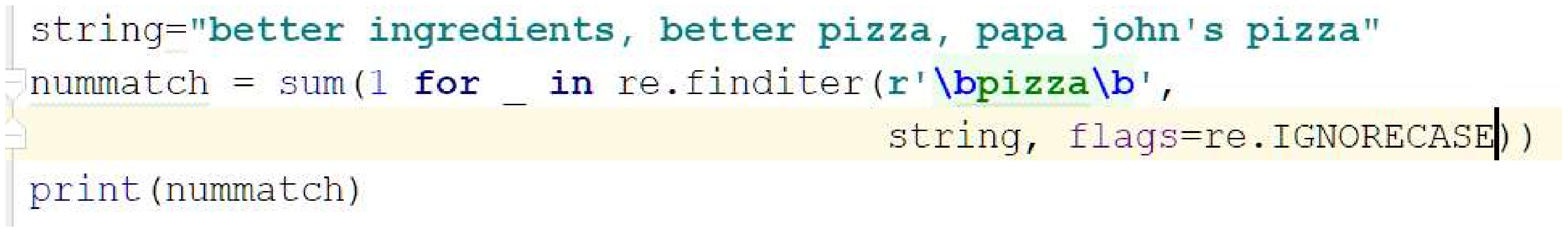


```
#\b - word boundary (a word is a sequence of word characters)
#\w - word characters, usually [a-zA-Z0-9_]
#* - match 0 or more of the preceding expression
#This finds any word containing "regulat"
regwords = re.findall(r'\b\w*regulat\w*\b', result, flags=re.IGNORECASE)
```

  - Need the \w* at the start to catch words like "deregulation"
  - Here, *regwords* will be a *list* with all matches
  - The "r" before the regular expression makes it a "raw" string, if this is not included the word boundary does not work correctly

Regular Expressions

- Counting the instances of a particular word with re.finditer
  - Setting up the regular expression:

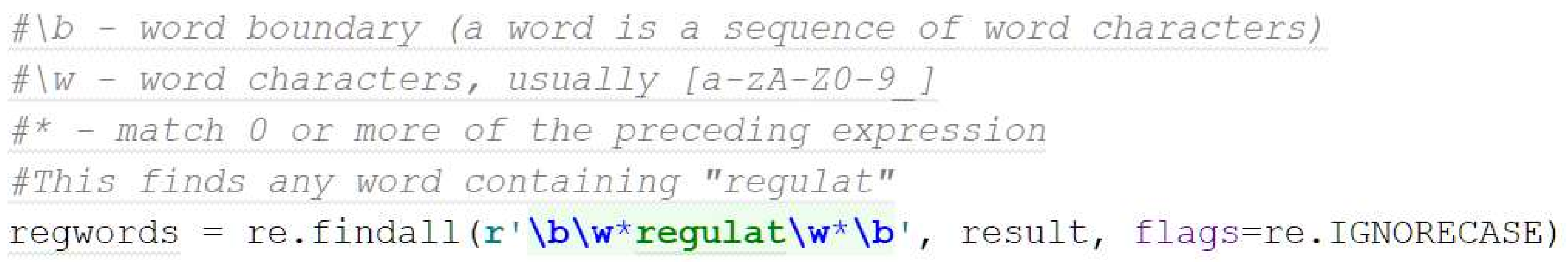


```
string="better ingredients, better pizza, papa john's pizza"
nummatch = sum(1 for _ in re.finditer(r'\bpizza\b',
                                        string, flags=re.IGNORECASE))
print(nummatch)
```

  - sum function – returns number of items iterated over
  - "_" – because we don't actually use the matches, "_" denotes a null argument
  - re.finditer loops over the matches
  - \b – word boundary

Regular Expressions

- Alternatively:



```
#\b - word boundary (a word is a sequence of word characters)
#\w - word characters, usually [a-zA-Z0-9_]
#* - match 0 or more of the preceding expression
#This finds any word containing "regulat"
regwords = re.findall(r'\b\w*regulat\w*\b', result, flags=re.IGNORECASE)
```

- Can use length of list to count the number of matches
  - *len(regwords)*

Counting the Number of Sentences that Contain a Specific String

- EX: Count the number of sentences about regulation
- Start with a string

```
string="I buy pizza ingredients. My dog likes bones. " \
       "Pizza is still unregulated. Dog bones are heavily regulated."
```

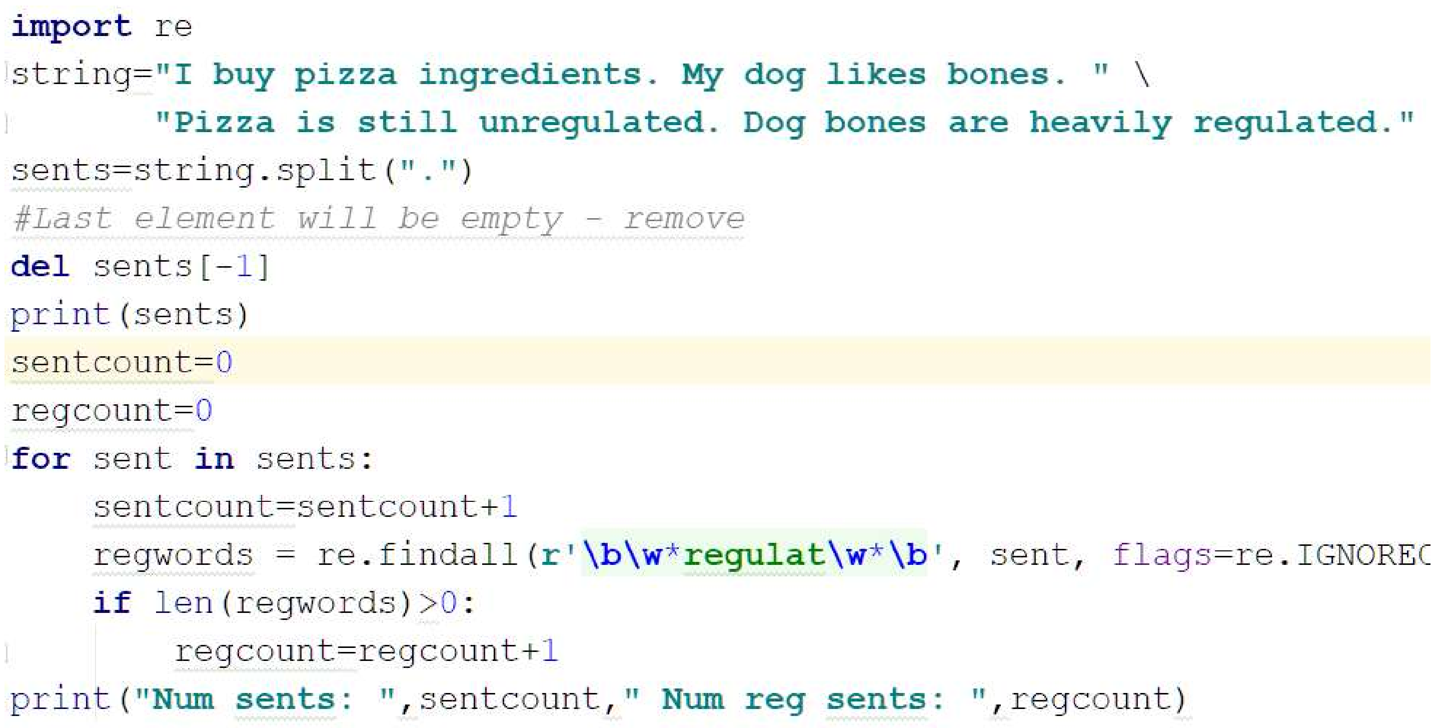
- Break into sentences using split

```
['I buy pizza ingredients', ' My dog likes bones', ' Pizza is still unregulated', ' Dog bones are heavily regulated']
Num sents:  4  Num reg sents:  2
```

- Then count the number of sentences that contain regulat*

Counting the Number of Sentences that Contain a Specific String

- Use split to break on “.”
    - Note – this may cause problems if document has strings like *“U.S.A.”*
- Use re.findall to identify matches

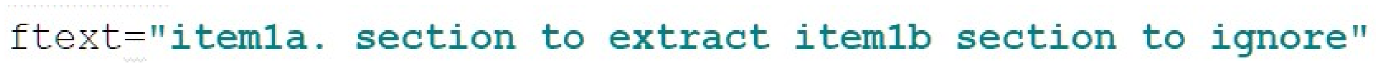


```
import re
string="I buy pizza ingredients. My dog likes bones. " \
       "Pizza is still unregulated. Dog bones are heavily regulated."
sents=string.split(".")
#Last element will be empty - remove
del sents[-1]
print(sents)
sentcount=0
regcount=0
for sent in sents:
    sentcount=sentcount+1
    regwords = re.findall(r'\b\w*regulat\w*\b', sent, flags=re.IGNOREC
    if len(regwords)>0:
        regcount=regcount+1
print("Num sents: ",sentcount," Num reg sents: ",regcount)
```

Extracting a text subsection

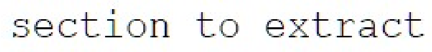
- Suppose you want to extract text between “item1a.” and “item1b”

Input:



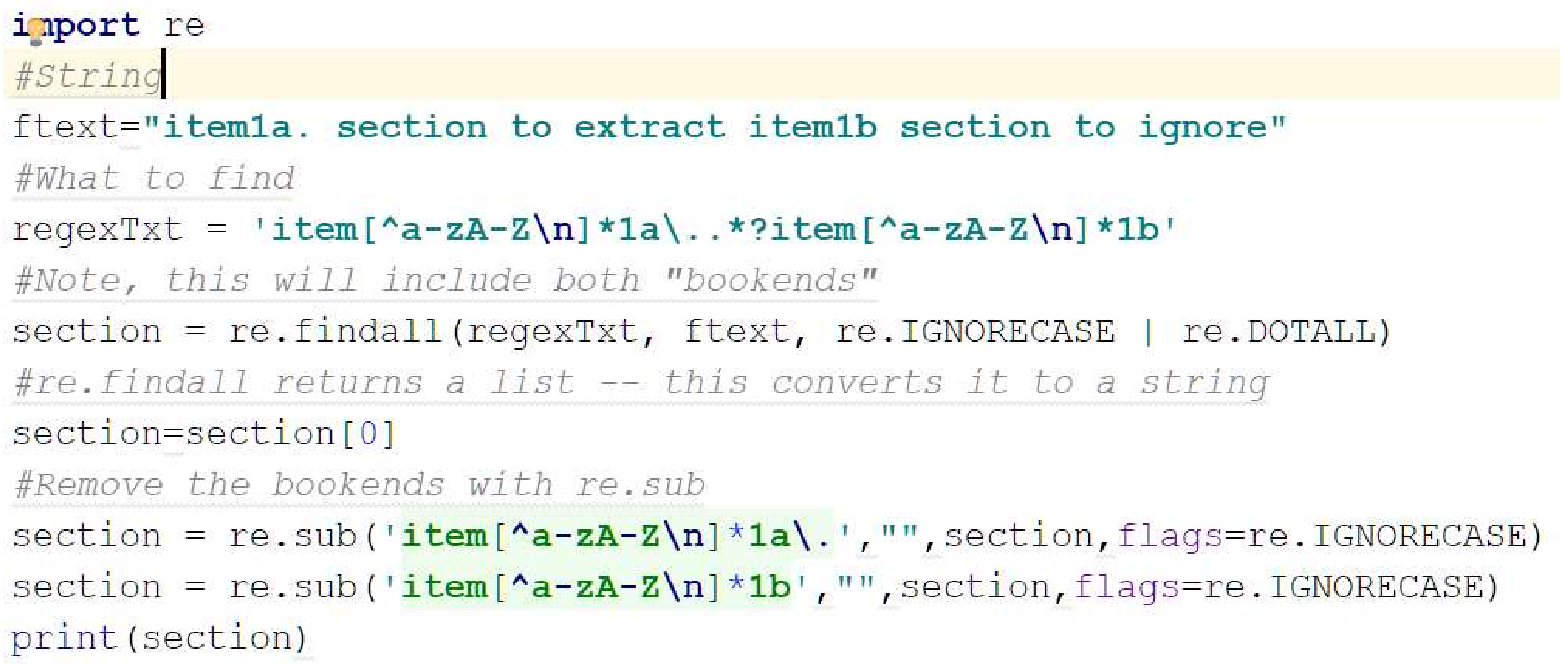
```
ftext="item1a. section to extract item1b section to ignore"
```

Desired Output:



```
section to extract
```

Extracting a string between a header and footer [Full Code/Output]



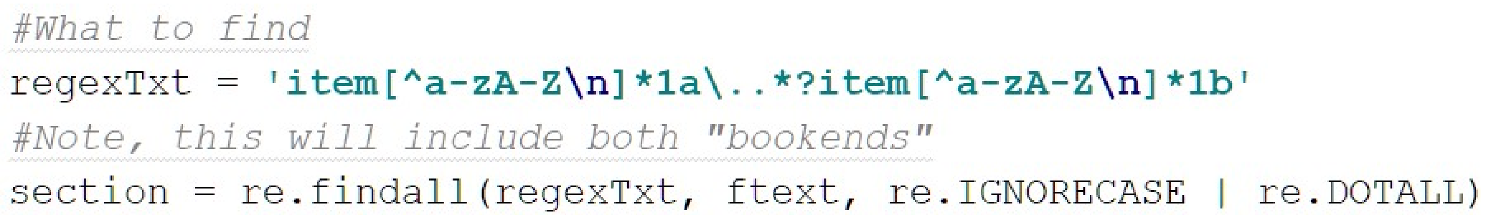
```python
import re
#String
ftext="item1a. section to extract item1b section to ignore"
#What to find
regexTxt = 'item[^a-zA-Z\n]*1a\..*?item[^a-zA-Z\n]*1b'
#Note, this will include both "bookends"
section = re.findall(regexTxt, ftext, re.IGNORECASE | re.DOTALL)
#re.findall returns a list -- this converts it to a string
section=section[0]
#Remove the bookends with re.sub
section = re.sub('item[^a-zA-Z\n]*1a\.',"",section,flags=re.IGNORECASE)
section = re.sub('item[^a-zA-Z\n]*1b',"",section,flags=re.IGNORECASE)
print(section)
```

Output:

```
section to extract
```

Extracting a string between a header and footer [Explanation]

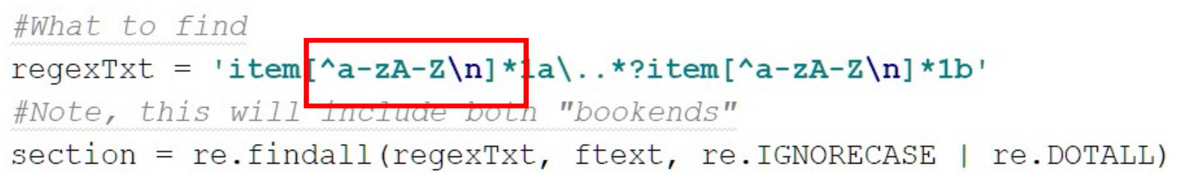
- Break down the regular expression
  - Caret (^) – This is a negation when used inside brackets. In this example, this matches any character, except a-z, A-Z or newline between “item” and “1a”
    - This will match item1a, item 1a, item.1a
    - Do not confuse with ^[a-zA-Z], which matches any string that starts with a letter, which is what the caret does outside of brackets
  - Dot (.) – match any character except a newline
    - Do not confuse with “\.” which matches a period



```
#What to find
regexTxt = 'item[^a-zA-Z\n]*1a\..*?item[^a-zA-Z\n]*1b'
#Note, this will include both "bookends"
section = re.findall(regexTxt, ftext, re.IGNORECASE | re.DOTALL)
```

Extracting a string between a header and footer [Explanation]

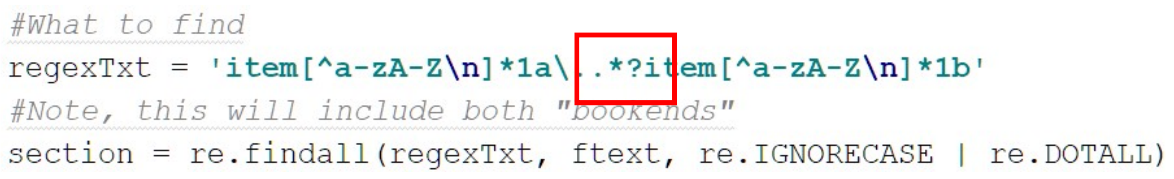
- Normally:
  - * matches zero or more of the preceding expression
    - In the boxed case, it would still be a match if it was written in the document as item1a, where there are no characters between the "item" and the "1a"
  - ? Matches zero or one of the preceding expression
  - the "." matches anything but a newline



```
#What to find
regexTxt = 'item[^a-zA-Z\n]*1a\..*?item[^a-zA-Z\n]*1b'
#Note, this will include both "bookends"
section = re.findall(regexTxt, ftext, re.IGNORECASE | re.DOTALL)
```

Extracting a string between a header and footer [Explanation]

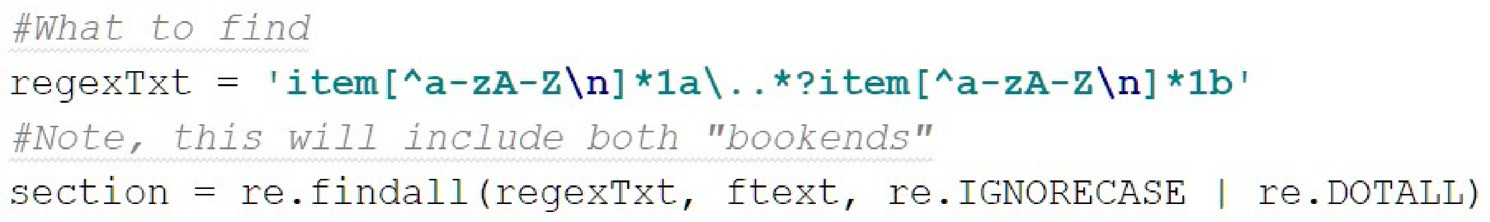
- Combining “*” and “?”
  - Example:
    - Input “101000000000100”
    - 1.*1 -> Matches 1010000000001 (greedy)
    - 1.*?1-> Matches 101 (reluctant)
  - For our section extraction, the boxed code will match first instance of item 1b appearing after item 1a



```
#What to find
regexTxt = 'item[^a-zA-Z\n]*1a\..*?item[^a-zA-Z\n]*1b'
#Note, this will include both "bookends"
section = re.findall(regexTxt, ftext, re.IGNORECASE | re.DOTALL)
```

Extracting a string between a header and footer [Explanation]

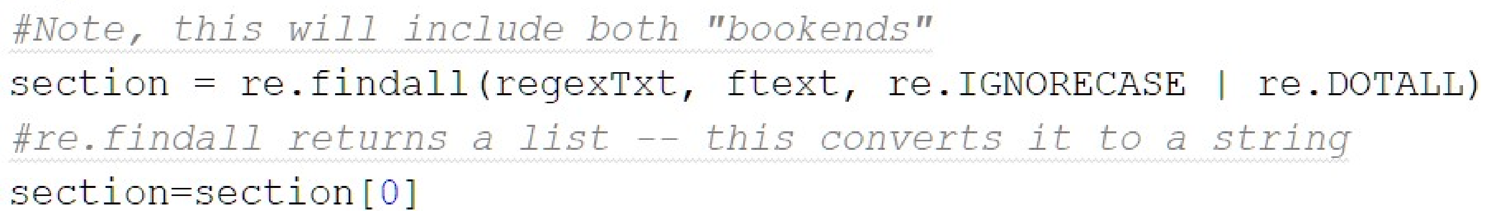
- Breaking Down the Regular Expressions:
  - IGNORECASE – do not require the desired string to both match characters and capitalization
  - DOTALL – normally, the "." matches anything but a newline, dotall allows it to match a newline character as well



```
#What to find
regexTxt = 'item[^a-zA-Z\n]*1a\..*?item[^a-zA-Z\n]*1b'
#Note, this will include both "bookends"
section = re.findall(regexTxt, ftext, re.IGNORECASE | re.DOTALL)
```

Extracting a string between a header and footer [Explanation]

- findall – puts the matches in a list, the main inputs are:
  - Regular expression to match
  - Where to look
- Note: You cannot perform string operations on a list, even if it has only one item

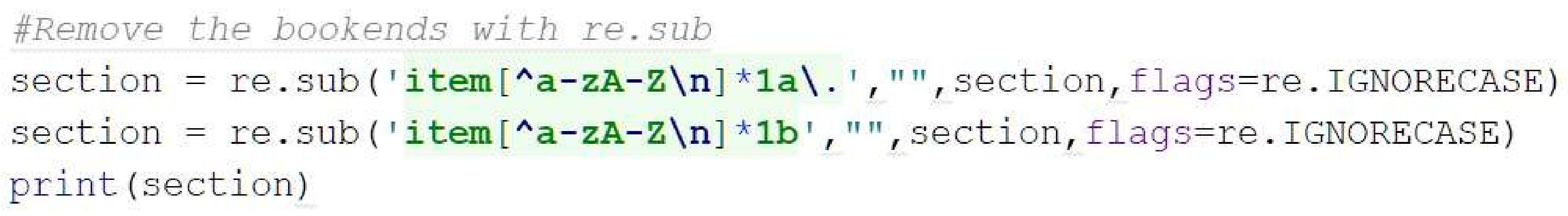


```
#Note, this will include both "bookends"
section = re.findall(regexTxt, ftext, re.IGNORECASE | re.DOTALL)
#re.findall returns a list -- this converts it to a string
section=section[0]
```

- There is only one match in this example, so we can use section[0] to extract it

Extracting a string between a header and footer [Explanation]

- Substituting text in a string
  - re.sub takes 3 main arguments
    - Item to find
    - What to replace it with
    - Where to look
- The code on the previous slides will also include item 1a and item 1b in the extracted string. We may want to remove these:



```
#Remove the bookends with re.sub
section = re.sub('item[^a-zA-Z\n]*1a\.',"",section,flags=re.IGNORECASE)
section = re.sub('item[^a-zA-Z\n]*1b',"",section,flags=re.IGNORECASE)
print(section)
```

Downloading SEC Files

Based on code at https://sraf.nd.edu/

Dlpython3v4.py

testsecdownload.py

- **IMPORTANT:** Only download from the SEC server in “off” hours
    - Opens at 9 PM EST
    - Closes at 6 AM EST
- Can implement with the following code:
    - Pass is needed for the code to run, otherwise you will get an indentation error

```
while EDGAR_Pac.edgar_server_not_available(True):
   pass
```

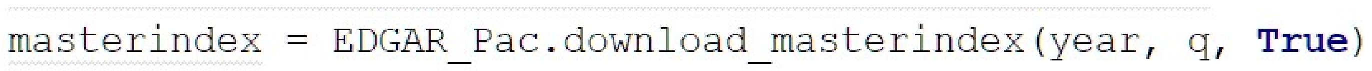
    - The *EDGAR_Pac.py* file will be posted on canvas, make sure to put it in the proper directory and import it at the top of your Python file

Downloading and Extracting Risk Factors

1. Download the master index
2. Loop over the elements (filings)
3. Extract the risk factors section
4. Save the extracted section

Downloading the Master Index

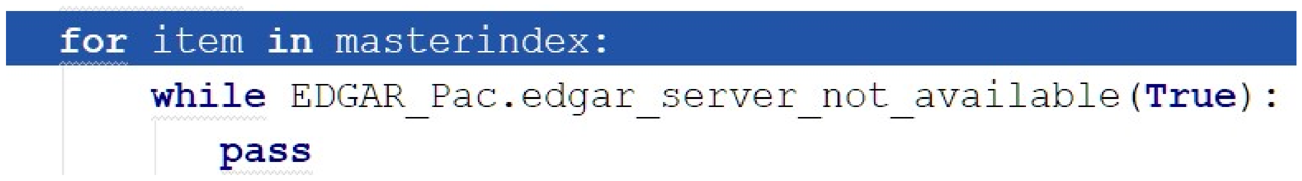
- Use *EDGAR_Pac.py* to download the master index
- The master index is a data structure where each item contains the cik, filing date, form type and firm name
    - There is a separate master index for each year and quarter (based on date the form was filed, not on the fiscal year/quarter end)



```
masterindex = EDGAR_Pac.download_masterindex(year, q, True)
```

Looping over the Master Index

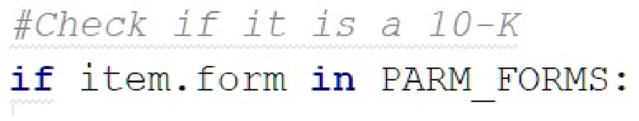
- Use the same syntax as you would for looping over a list
- **NOTE: This code will only run between 9PM EST and 6AM EST**



```python
for item in masterindex:
    while EDGAR_Pac.edgar_server_not_available(True):
        pass
```

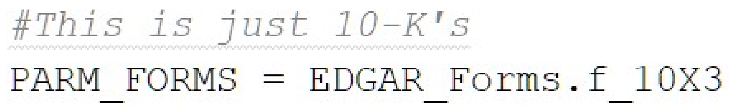
Checking for the Correct File Type

- There are tons of filings, and only a small fraction are 10-K’s
- In the loop, check to make sure the file is a 10-K to save time and storage space



```
#Check if it is a 10-K
if item.form in PARM_FORMS:
```

- Note 1: Each item in masterindex item has several elements. To refer to an element, use the syntax *item.your_desired_element*
- Note 2: I previously set PARM_FORMS to only include “10-K”s



```
#This is just 10-K's
PARM_FORMS = EDGAR_Forms.f_10X3
```

Extracting the Risk Factors Section

- Recall the example of extracting a section between two “bookends”
- The bookends might appear in multiple places
  - The most common “duplicate” is in table of contents, and body of the document.
  - Remember that re.search returns a list of strings. To select the longest element (which is usually the true risk factors section) use:

```
result = max(section,key=len)
```

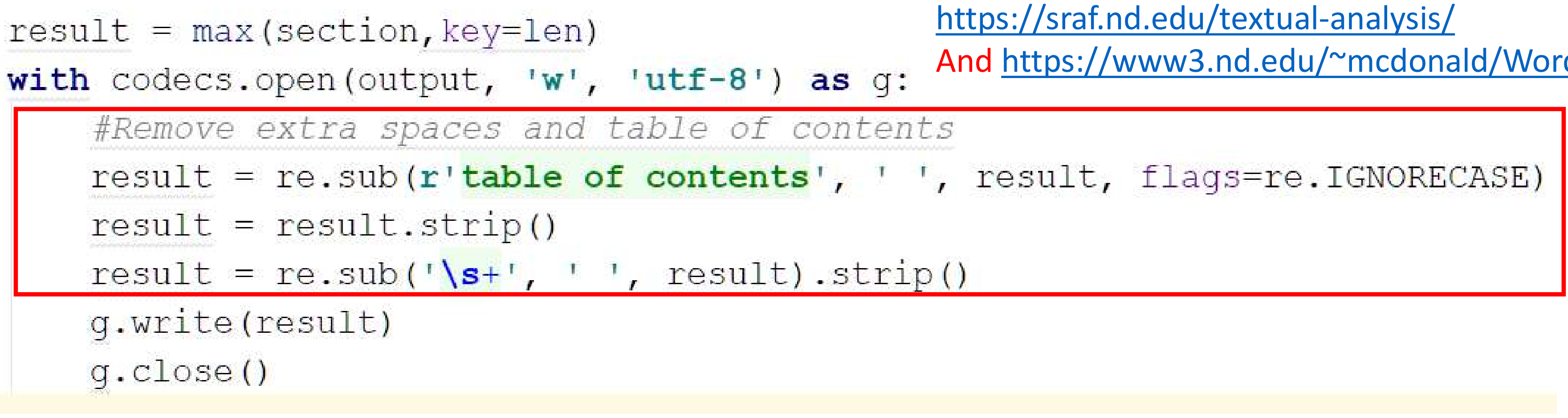
Exporting the Risk Factors Section

- Make sure you choose a unique filename, I like to use the CIK (firm id), the date the form was filed, and the form type

```
output = './testfiles2/' + str(year) + '/' + str(item.cik) \
         + '-' + str(item.filingdate) + '-' + item.form + '.txt'
```

- Then use codecs.open

For more detailed cleaning, see https://sraf.nd.edu/textual-analysis/
And https://www3.nd.edu/~mcdonald/Word_Lists.html



```
result = max(section,key=len)
with codecs.open(output, 'w', 'utf-8') as g:
    #Remove extra spaces and table of contents
    result = re.sub(r'table of contents', ' ', result, flags=re.IGNORECASE)
    result = result.strip()
    result = re.sub('\s+', ' ', result).strip()
    g.write(result)
    g.close()
```

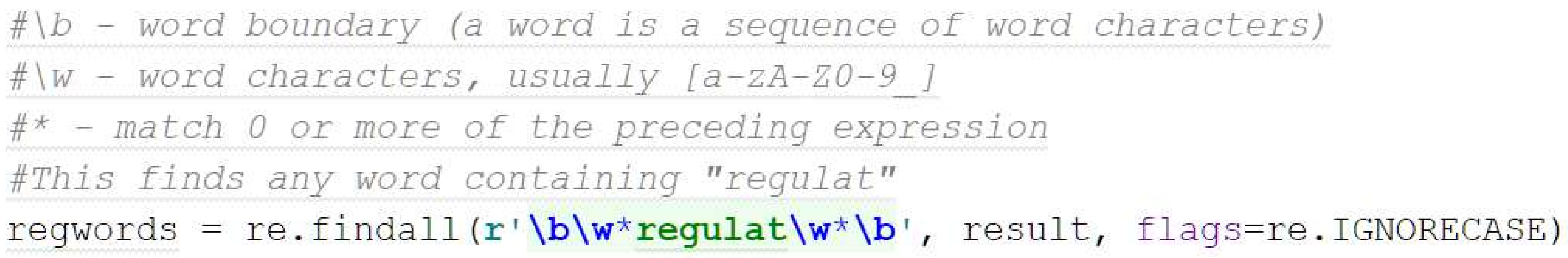
Collecting Document Characteristics

Scan_detail.py

Dataframes.py

Counting occurrences of “Regulation” scan_detail.py

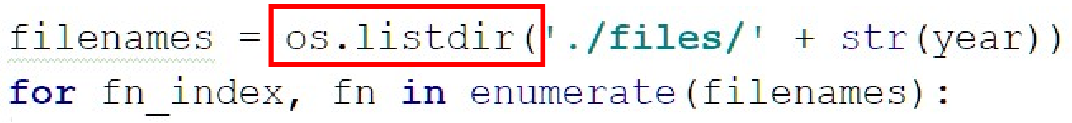
- After downloading all the risk factors sections, want to extract counts of particular words
- Each row will be the counts for a particular document
- You can use all the regular expressions discussed previously



```
#\b - word boundary (a word is a sequence of word characters)
#\w - word characters, usually [a-zA-Z0-9_]
#* - match 0 or more of the preceding expression
#This finds any word containing "regulat"
regwords = re.findall(r'\b\w*regulat\w*\b', result, flags=re.IGNORECASE)
```

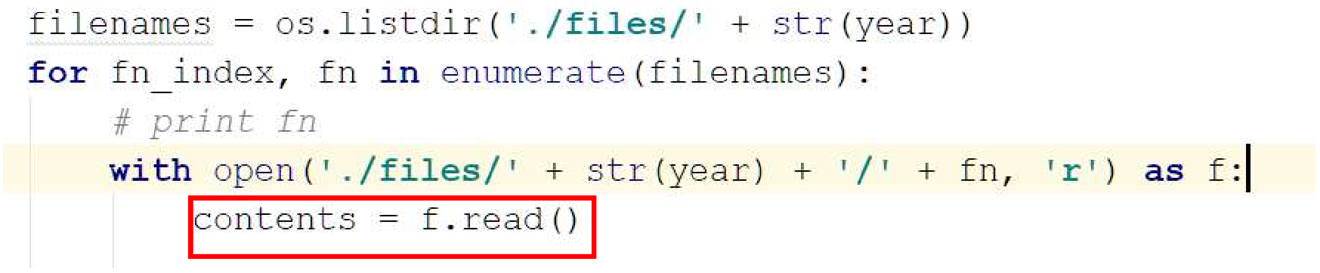
Finding the files

- Looping over files in a directory
  - os.listdir creates a list with all the files in a particular directory
    - *Remember to import os*
  - Here, fn_index is the file number, while fn is the file name
    - Note – the enumerate function adds the counter "fn_index" to the iterable "fn"



```
filenames = os.listdir('./files/' + str(year))
for fn_index, fn in enumerate(filenames):
```

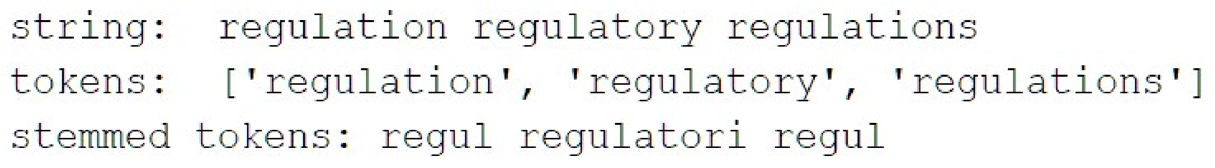
  - Import the file as a string, and all the regex described previously will work



```
filenames = os.listdir('./files/' + str(year))
for fn_index, fn in enumerate(filenames):
    # print fn
    with open('./files/' + str(year) + '/' + fn, 'r') as f:
        contents = f.read()
```

Dealing with words vs. word stems

- Want to match: regulation, regulatory and regulations
- You can use the complicated regular expression on the previous slide, or you can stem the words

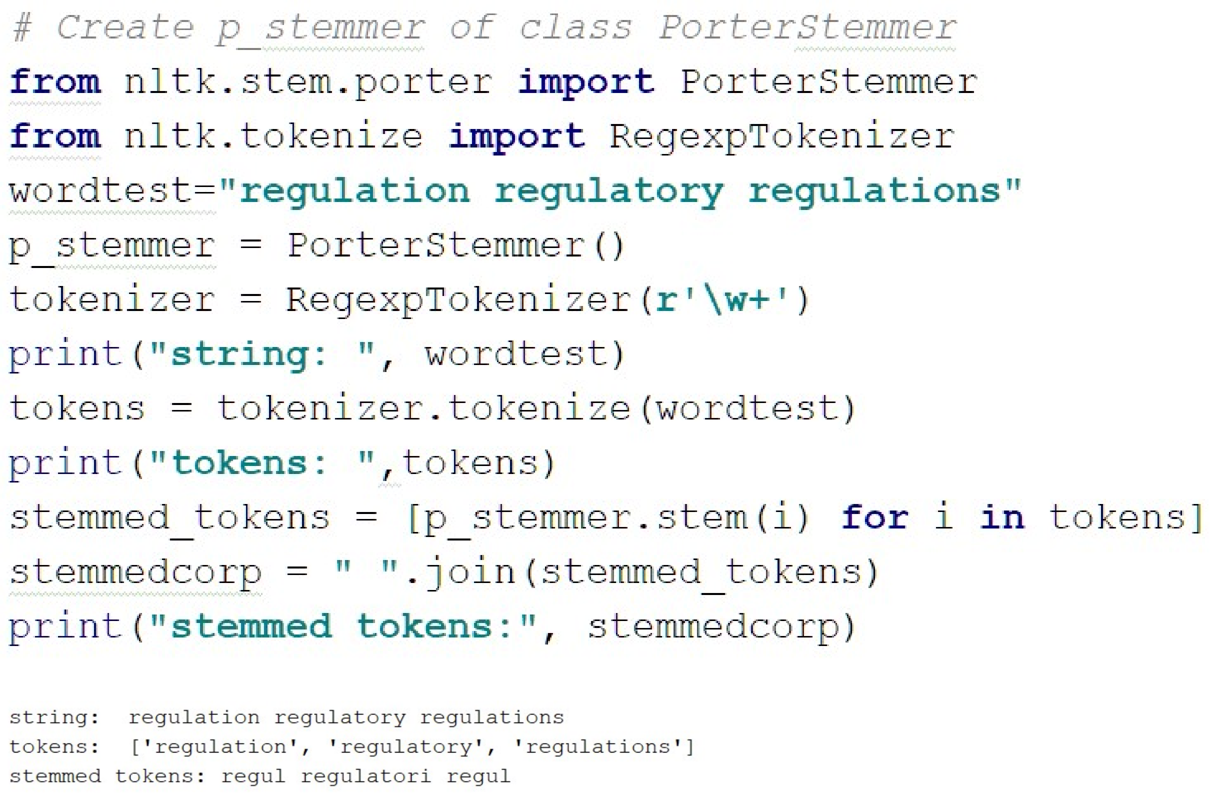


```
string:  regulation regulatory regulations
tokens:  ['regulation', 'regulatory', 'regulations']
stemmed tokens: regul regulatori regul
```

- Note – stemming is time consuming, so it may be faster to just use regex
    - Bonus – use re.compile for even more speed

Dealing with words vs. word stems

- Here, tokenizer identifies “words”, puts them in a list
  - Note, the + matches *1 or more*
- Porter stemmer finds the “stem” of each item in the list
- I put the list back together with *“ “.join()*
  - Join appends the elements of a list, putting the character before the join between each element



```
# Create p_stemmer of class PorterStemmer
from nltk.stem.porter import PorterStemmer
from nltk.tokenize import RegexpTokenizer
wordtest="regulation regulatory regulations"
p_stemmer = PorterStemmer()
tokenizer = RegexpTokenizer(r'\w+')
print("string: ", wordtest)
tokens = tokenizer.tokenize(wordtest)
print("tokens: ",tokens)
stemmed_tokens = [p_stemmer.stem(i) for i in tokens]
stemmedcorp = " ".join(stemmed_tokens)
print("stemmed tokens:", stemmedcorp)
```

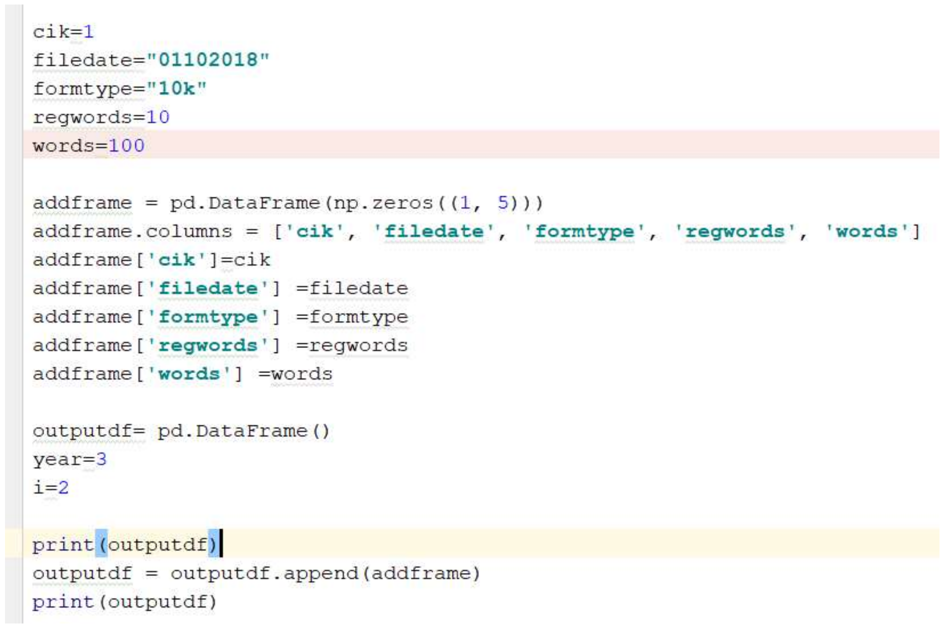
```
string:  regulation regulatory regulations
tokens:  ['regulation', 'regulatory', 'regulations']
stemmed tokens: regul regulatori regul
```

How to Store the Extracted Data

- scan_detail.py writes the output to a csv file one line at a time
- An alternative is to use pandas, create a “Data Frame”, and then write the data frame to a csv
  - Remember to call pandas with: *import pandas as pd*

How to access elements of a data frame

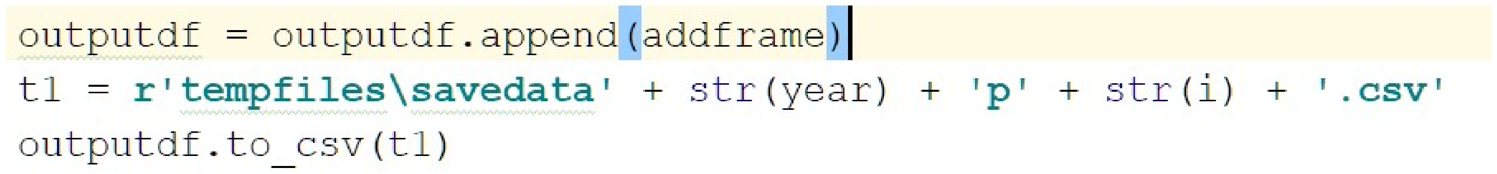
- I like to label the columns, and use the syntax:
  - *Your_data_frame['your_column']*



```
cik=1
filedate="01102018"
formtype="10k"
regwords=10
words=100

addframe = pd.DataFrame(np.zeros((1, 5)))
addframe.columns = ['cik', 'filedate', 'formtype', 'regwords', 'words']
addframe['cik']=cik
addframe['filedate'] =filedate
addframe['formtype'] =formtype
addframe['regwords'] =regwords
addframe['words'] =words

outputdf= pd.DataFrame()
year=3
i=2

print(outputdf)
outputdf = outputdf.append(addframe)
print(outputdf)
```

Appending data frames to data frames

- Because you are looping over files, it may not be possible to write the whole data frame in one shot
- Solution: At each step in the loop, append the smaller data frame (ex. a data frame with just one row/observation) to a master data frame, and export it to a csv



```
outputdf = outputdf.append(addframe)
t1 = r'tempfiles\savedata' + str(year) + 'p' + str(i) + '.csv'
outputdf.to_csv(t1)
```

- Be careful when trying to parallelize loops

Financial Health Economics by Koijen et. al. (2016)

http://onlinelibrary.wiley.com/doi/10.3982/ECTA11182/abstract

Summary of Results

- Firms investing in medical innovation earn 4%-6% more than predicted by standard asset pricing models
- The authors propose this is compensation for government-induced profit risk
- To measure this, they look at word frequencies in 10-K risk factors:
    1. Healthcare firms discuss government-related risks more than average
    2. Returns are negative going forward when there are threats of government intervention
    3. The medical firms with the biggest stock losses during the Clinton healthcare reforms were more exposed to a “medical innovation factor”

Dictionary

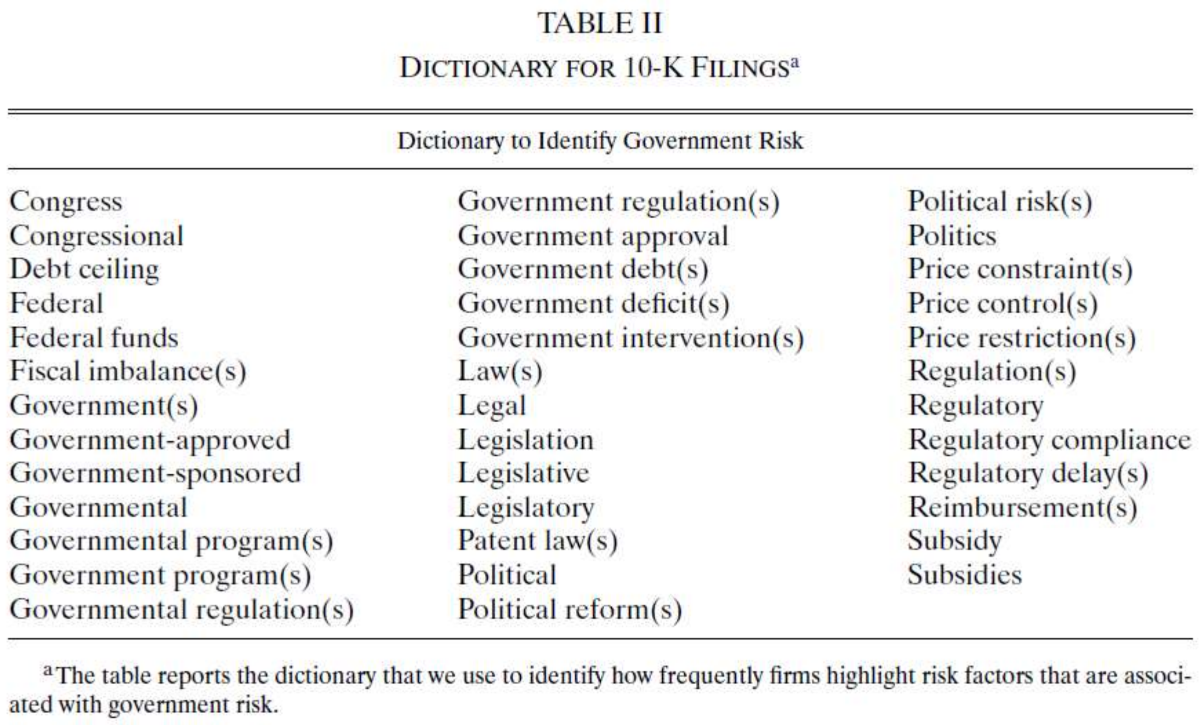


TABLE II

DICTIONARY FOR 10-K FILINGS[a]

Dictionary to Identify Government Risk		
Congress	Government regulation(s)	Political risk(s)
Congressional	Government approval	Politics
Debt ceiling	Government debt(s)	Price constraint(s)
Federal	Government deficit(s)	Price control(s)
Federal funds	Government intervention(s)	Price restriction(s)
Fiscal imbalance(s)	Law(s)	Regulation(s)
Government(s)	Legal	Regulatory
Government-approved	Legislation	Regulatory compliance
Government-sponsored	Legislative	Regulatory delay(s)
Governmental	Legislatory	Reimbursement(s)
Governmental program(s)	Patent law(s)	Subsidy
Government program(s)	Political	Subsidies
Governmental regulation(s)	Political reform(s)	

[a] The table reports the dictionary that we use to identify how frequently firms highlight risk factors that are associated with government risk.

Words vs. n-Grams

- The dictionary has both words and n-grams
- Some of the 2-grams contain standalone words
  - Example: “political” and “political reforms”
- To get an accurate count:
  - Count the number of n-grams and words
  - Add together the counts, then subtract the # of n-gram’s containing standalone words, times the number of standalone words in that n-gram

Words vs. n-Grams

- Example: *political* occurs 8 times, *regulation* occurs 4 times, *political risk* occurs 2 times and *political regulation* occurs 3 times.
  - Count the number of n-grams and words

Word/2-Gram	# Appearances
Political	8
Regulation	4
Political risk	2
Political regulation	3

  - Total Word Count:
    - Total matches –> 17
    - Subtract overlap –> 2 from political risk, 6 from political regulation
    - Final –> 17-8 = **9**

Details

1. Healthcare firms discuss government-related risks more than average
   - Words from their dictionary appear 130 times on average for healthcare firms, and 77 times for non-healthcare firms
2. Returns are negative going forward when there are threats of government intervention
   - Drawdown during Clinton reforms, not during ACA
     - Clinton reforms included price controls
3. The medical firms that were hurt the most during the Clinton healthcare reforms were more exposed to a "medical innovation factor"
   - Run a bivariate regression on mkt and healthcare industry, those with a higher healthcare beta are more exposed to "medical innovation"