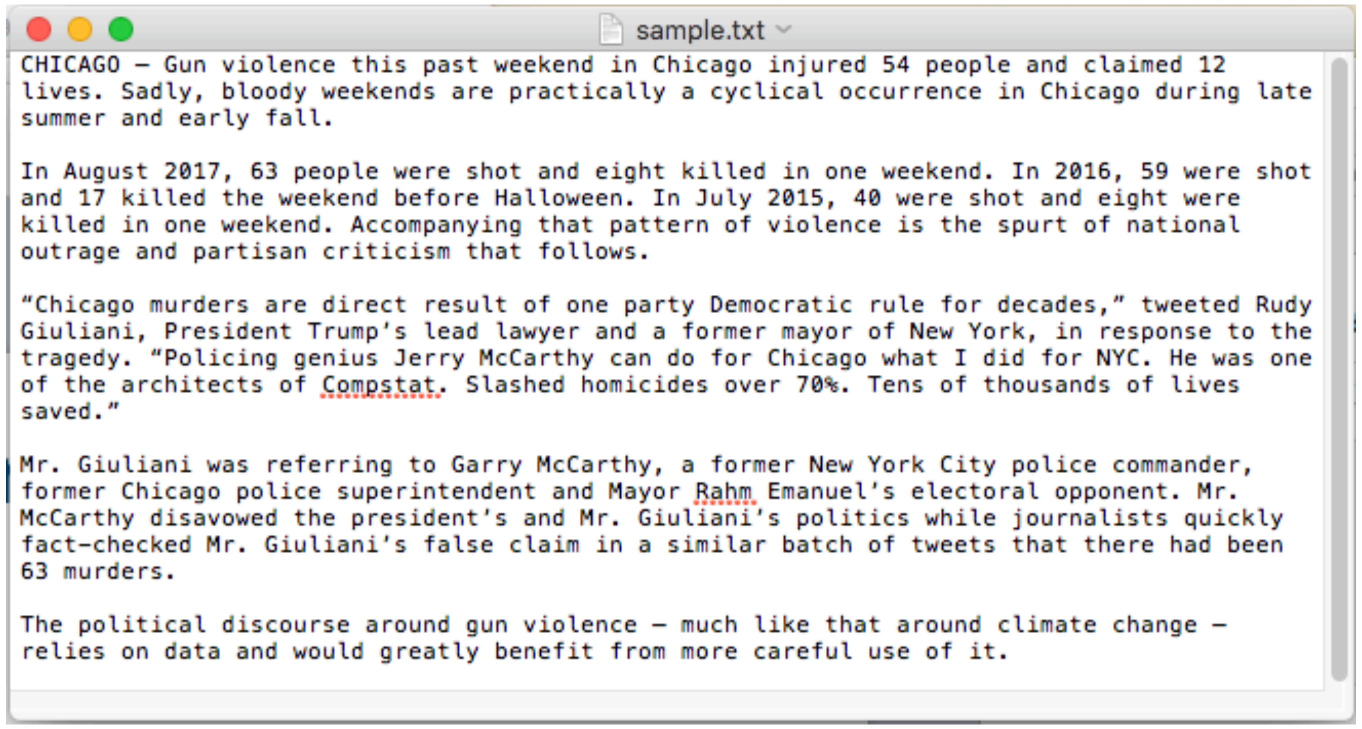
A REFRESHER — IDE (Atom, Sublime, etc)

- Atom, Sublime, etc are **not** Python-specific
- They are programs just like any other on our computer
- They are text editors (like Microsoft Word, Text Edit, Notepad), which allow us to display and edit text
- However, they're catered for the task of **programming**, so they have nifty features like displaying certain words w/ colors to make it easy to read
- Related, we could write Python code in Microsoft Word if we wanted, just that would make our life more difficult

Opening a NY Times article



CHICAGO — Gun violence this past weekend in Chicago injured 54 people and claimed 12 lives. Sadly, bloody weekends are practically a cyclical occurrence in Chicago during late summer and early fall.

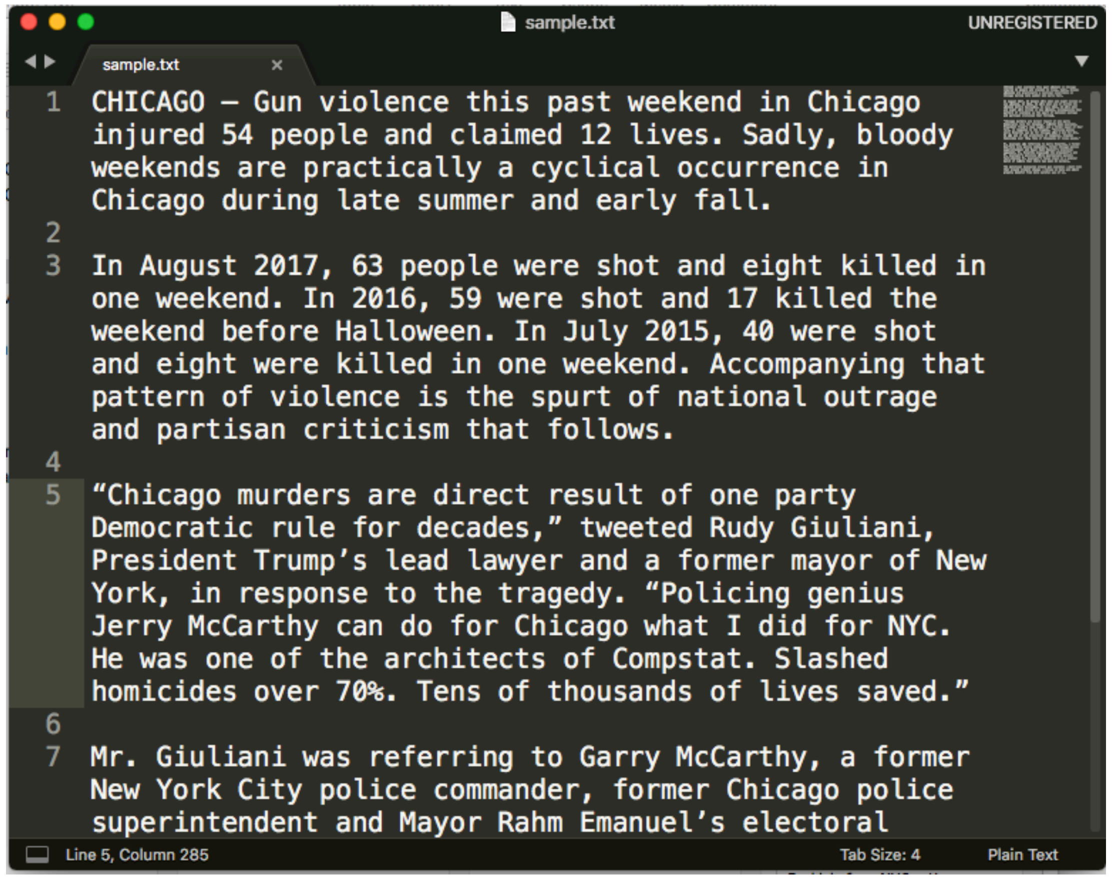
In August 2017, 63 people were shot and eight killed in one weekend. In 2016, 59 were shot and 17 killed the weekend before Halloween. In July 2015, 40 were shot and eight were killed in one weekend. Accompanying that pattern of violence is the spurt of national outrage and partisan criticism that follows.

“Chicago murders are direct result of one party Democratic rule for decades,” tweeted Rudy Giuliani, President Trump’s lead lawyer and a former mayor of New York, in response to the tragedy. “Policing genius Jerry McCarthy can do for Chicago what I did for NYC. He was one of the architects of Compstat. Slashed homicides over 70%. Tens of thousands of lives saved.”

Mr. Giuliani was referring to Garry McCarthy, a former New York City police commander, former Chicago police superintendent and Mayor Rahm Emanuel’s electoral opponent. Mr. McCarthy disavowed the president’s and Mr. Giuliani’s politics while journalists quickly fact-checked Mr. Giuliani’s false claim in a similar batch of tweets that there had been 63 murders.

The political discourse around gun violence — much like that around climate change — relies on data and would greatly benefit from more careful use of it.

Text Edit



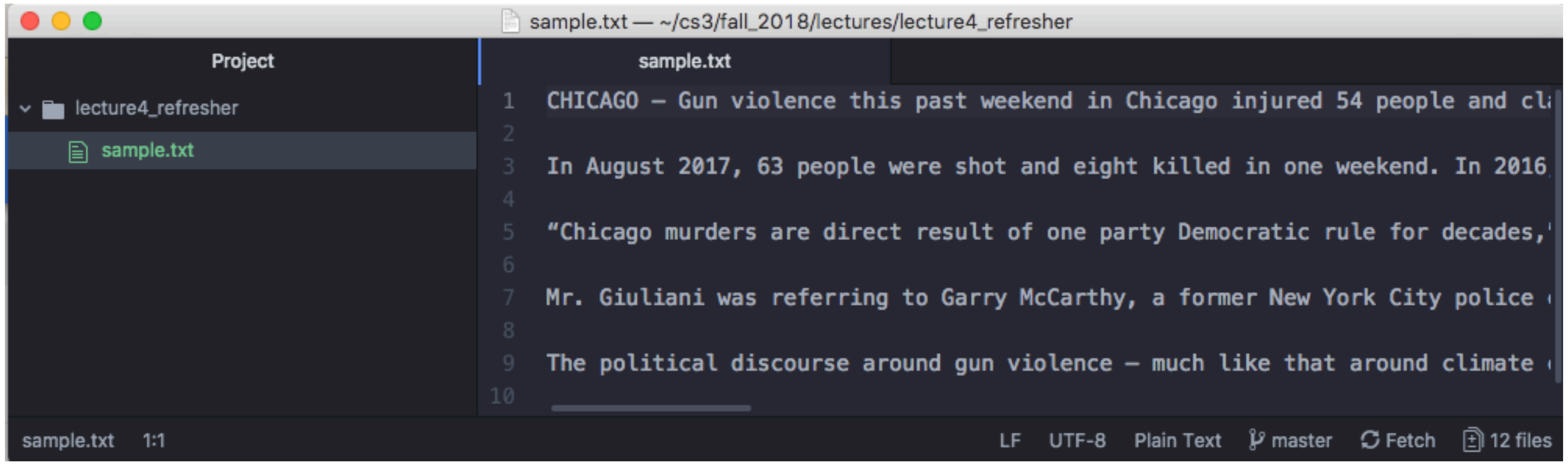
CHICAGO — Gun violence this past weekend in Chicago injured 54 people and claimed 12 lives. Sadly, bloody weekends are practically a cyclical occurrence in Chicago during late summer and early fall.

In August 2017, 63 people were shot and eight killed in one weekend. In 2016, 59 were shot and 17 killed the weekend before Halloween. In July 2015, 40 were shot and eight were killed in one weekend. Accompanying that pattern of violence is the spurt of national outrage and partisan criticism that follows.

“Chicago murders are direct result of one party Democratic rule for decades,” tweeted Rudy Giuliani, President Trump’s lead lawyer and a former mayor of New York, in response to the tragedy. “Policing genius Jerry McCarthy can do for Chicago what I did for NYC. He was one of the architects of Compstat. Slashed homicides over 70%. Tens of thousands of lives saved.”

Mr. Giuliani was referring to Garry McCarthy, a former New York City police commander, former Chicago police superintendent and Mayor Rahm Emanuel’s electoral

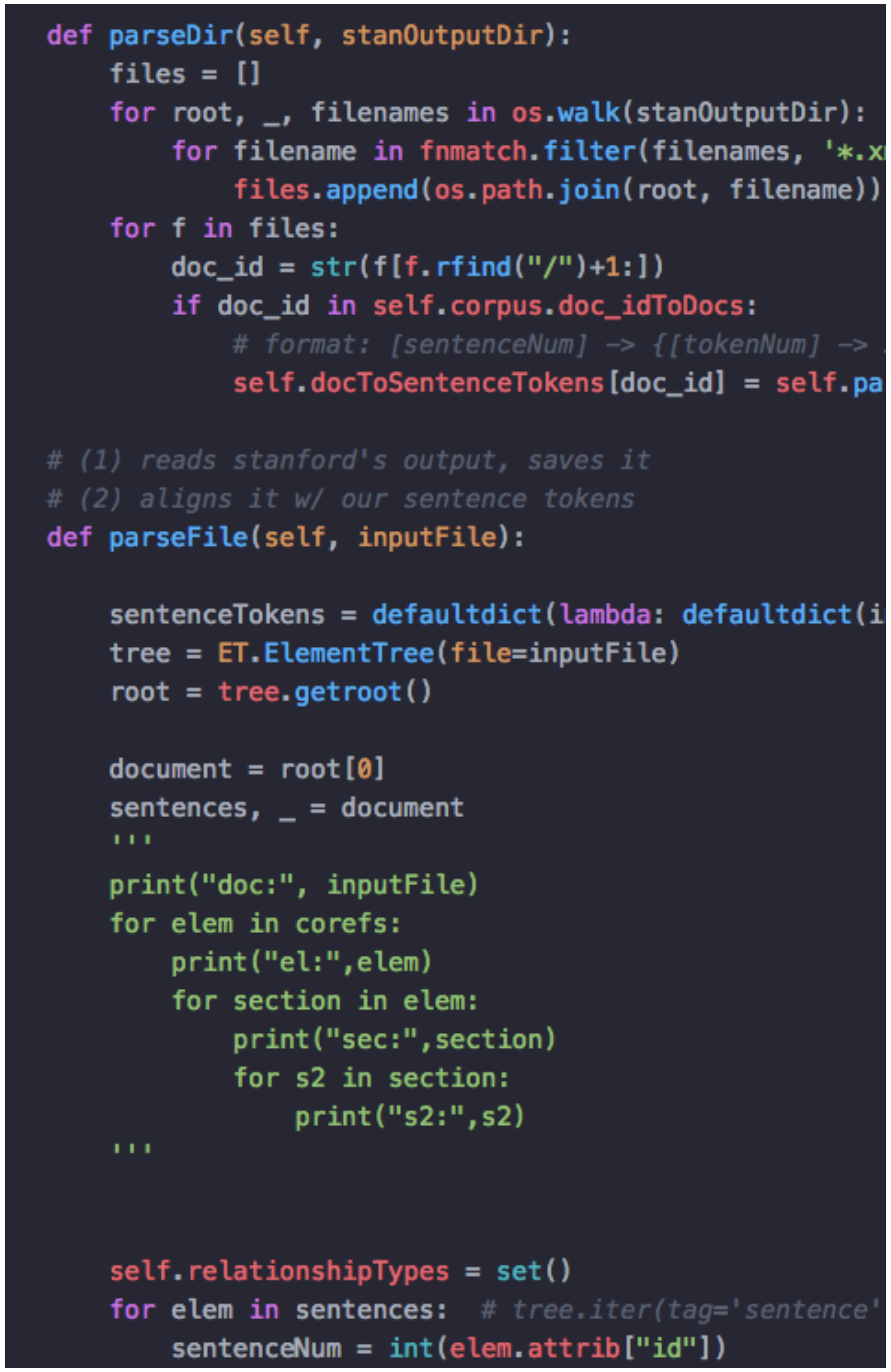
Sublime



```
CHICAGO — Gun violence this past weekend in Chicago injured 54 people and cl

In August 2017, 63 people were shot and eight killed in one weekend. In 2016

“Chicago murders are direct result of one party Democratic rule for decades,

Mr. Giuliani was referring to Garry McCarthy, a former New York City police

The political discourse around gun violence — much like that around climate
```

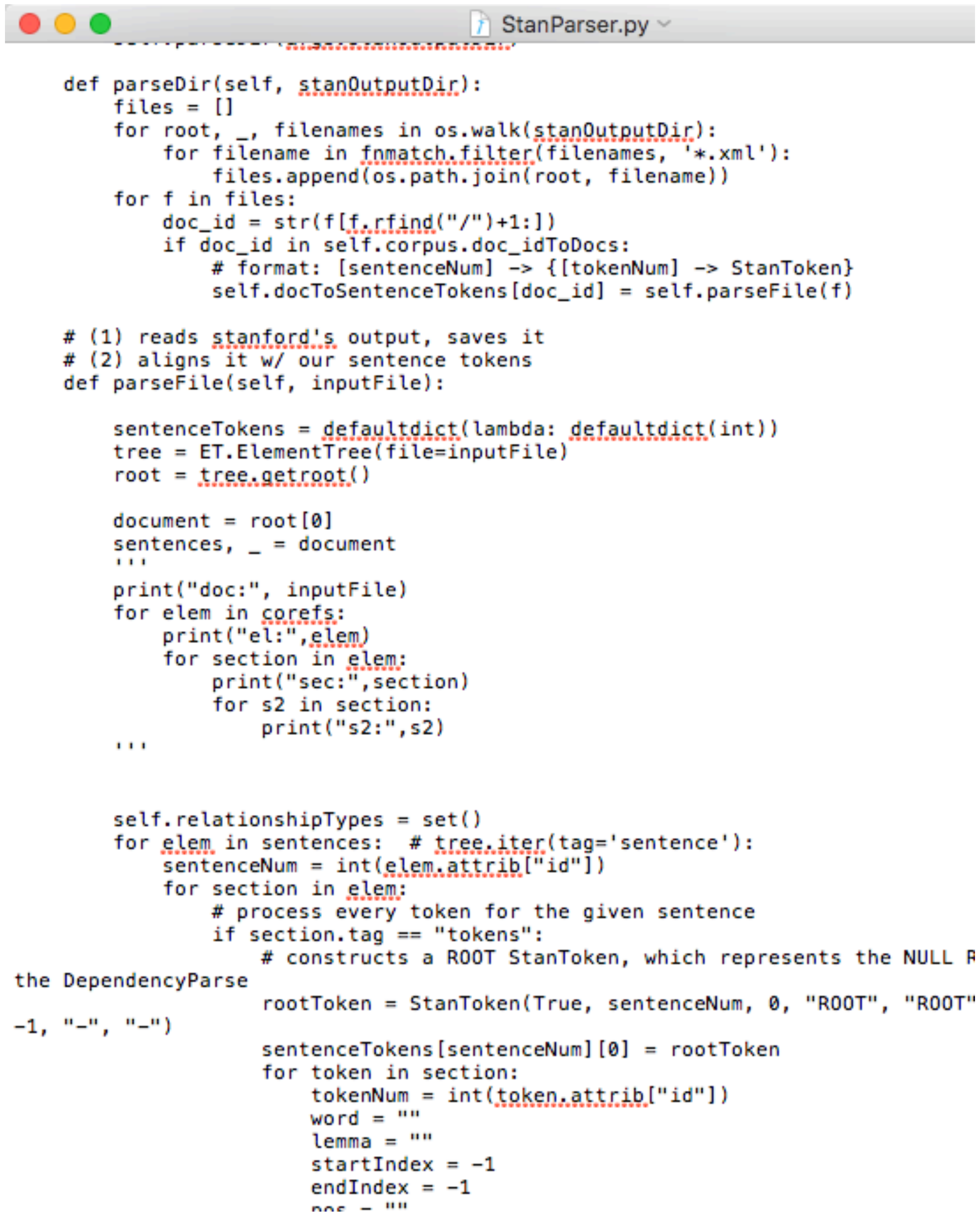
Atom

Opening a file of Python code



```
def parseDir(self, stanOutputDir):
    files = []
    for root, _, filenames in os.walk(stanOutputDir):
        for filename in fnmatch.filter(filenames, '*.x
            files.append(os.path.join(root, filename))
    for f in files:
        doc_id = str(f[f.rfind("/")+1:])
        if doc_id in self.corpus.doc_idToDocs:
            # format: [sentenceNum] -> {[tokenNum] ->
            self.docToSentenceTokens[doc_id] = self.pa

# (1) reads stanford's output, saves it
# (2) aligns it w/ our sentence tokens
def parseFile(self, inputFile):

    sentenceTokens = defaultdict(lambda: defaultdict(i
    tree = ET.ElementTree(file=inputFile)
    root = tree.getroot()

    document = root[0]
    sentences, _ = document
    '''
    print("doc:", inputFile)
    for elem in corefs:
        print("el:",elem)
        for section in elem:
            print("sec:",section)
            for s2 in section:
                print("s2:",s2)
    '''


    self.relationshipTypes = set()
    for elem in sentences:  # tree.iter(tag='sentence'
        sentenceNum = int(elem.attrib["id"])
```

Atom



```
    def parseDir(self, stanOutputDir):
        files = []
        for root, _, filenames in os.walk(stanOutputDir):
            for filename in fnmatch.filter(filenames, '*.xml'):
                files.append(os.path.join(root, filename))
        for f in files:
            doc_id = str(f[f.rfind("/")+1:])
            if doc_id in self.corpus.doc_idToDocs:
                # format: [sentenceNum] -> {[tokenNum] -> StanToken}
                self.docToSentenceTokens[doc_id] = self.parseFile(f)

    # (1) reads stanford's output, saves it
    # (2) aligns it w/ our sentence tokens
    def parseFile(self, inputFile):

        sentenceTokens = defaultdict(lambda: defaultdict(int))
        tree = ET.ElementTree(file=inputFile)
        root = tree.getroot()

        document = root[0]
        sentences, _ = document
        '''
        print("doc:", inputFile)
        for elem in corefs:
            print("el:",elem)
            for section in elem:
                print("sec:",section)
                for s2 in section:
                    print("s2:",s2)
        '''


        self.relationshipTypes = set()
        for elem in sentences:  # tree.iter(tag='sentence'):
            sentenceNum = int(elem.attrib["id"])
            for section in elem:
                # process every token for the given sentence
                if section.tag == "tokens":
                    # constructs a ROOT StanToken, which represents the NULL R
the DependencyParse
                    rootToken = StanToken(True, sentenceNum, 0, "ROOT", "ROOT"
-1, "-", "-")
                    sentenceTokens[sentenceNum][0] = rootToken
                    for token in section:
                        tokenNum = int(token.attrib["id"])
                        word = ""
                        lemma = ""
                        startIndex = -1
                        endIndex = -1
```

Text Edit

A REFRESHER — Python

- We choose to teach programming via the Python language
- We chose Python because:
    - it's incredibly powerful (arguably the most robust language)
    - easy to read and write code
    - extensive set of **libraries** to help w/ doing technical stuff
- The skills you learn in this course (including writing Python code) are completely transferrable to other programming languages; after this course, it would be easy to write code in Java or R (just as Caroline, the TA. She took CS3). The core principles are the same!

A REFRESHER — Anaconda

- Anaconda is just the software that helps us install Python on our computers
- We didn’t have to use it; there are other ways to install Python, but it’s generally a very easy way to install Python

A REFRESHER — The Terminal (aka Console)

- The terminal/console isn't Python-specific! Inherently, has nothing to do with Python

- It merely provides an alternative way to function w/ our computer, instead of the normal, graphical way with our mouse and clicking on folders and double-clicking programs to open them

- Instead of using your computer via a mouse and clicking on pretty things, one could do most things while just using the Terminal/Console

- Our Python programs we'll create in this course don't have graphical components that display stuff on the screen (e.g., Spotify), so it makes most sense to execute them from the Terminal