Appendix 2

main.py

```
#!/usr/local/bin/python3
#
# Paul Evans (10evans@cua.edu)
# 8 February 2015 -
# 12 February 2015
#
import re
import sys
import parse
def main():
    file = open('./edF.txt', 'r').read()
    decretum = parse.parse_all(preprocess(file))
    traverse(decretum[0])
    traverse(decretum[1])
    traverse(decretum[2])

def traverse(tree):
    for i in range(len(tree[1])):
        subtree = tree[1][i]
        if isinstance(subtree[1], list):
            tag = subtree[0]
            print(tag)
            traverse(subtree)
        elif isinstance(subtree[1], str):
            tag = subtree[0]
```

```
            text = subtree[1]
            print(tag, text)
    return

def preprocess(text):
    text = re.sub(re.compile('\-.*?\+', re.S), '', text) # remove comments
    text = re.sub('\<S \d{1,4}\>', '', text) # remove page number tags
    text = re.sub('\<L \d{1,2}\>', '', text) # remove line number tags
    text = re.sub('\<P 1\>|\<P 0\>', '', text) # remove Palea tags
    text = re.sub('\s+', ' ', text) # remove multiple whitespace characters
    text = re.sub('\s+$', '', text) # remove trailing whitespace characters
    return(text)

if __name__ == '__main__':
    main()
```

parse.py

```
#!/usr/local/bin/python3
#
# Paul Evans (10evans@cua.edu)
# 23 January 2015 -
# 12 February 2015
#
import re
import sys
def parse_all(text):
    part_list = []
    m = re.search('(\<1 D\>.*?)(\<1 C\>.*?)(\<1 DC\>.*?)$', text, re.S)
    part_list.append(('<1 D>', parse_part_1(m.group(1))))
    part_list.append(('<1 C>', parse_part_2(m.group(2))))
    part_list.append(('<1 DC>', parse_part_3(m.group(3))))
```

```
    return(part_list)

# D.1-101
def parse_part_1(text):
    distinction_list = []
    distinctions = re.findall('(?:\<1 D\>)(.*?)(?=\<1 D\>|$)', text)
    for distinction in distinctions:
        distinction = distinction.strip(' ')
        m = re.match('(\<2 \d{1,3}\>) (\<T A\>) (.*?) (\<4 1\>.*?)$', distinction)
        tag = m.group(1)
        node = (m.group(2), m.group(3)) # d.a.c.1 tag-text tuple
        canon_list = parse_canons(m.group(4))
        canon_list.insert(0, node)
        distinction_list.append((tag, canon_list))
    return(distinction_list)

# C.1-36
def parse_part_2(text):
    case_list = []
    cases = re.findall('(?:\<1 C\>)(.*?)(?=\<1 C\>|$)', text)
    for case in cases:
        case = case.strip(' ')
        m = re.match('(\<2 \d{1,2}\>)(\<T Q\>) (.*?) (\<3 1\>.*?)$', case)
        tag = m.group(1)
        node = (m.group(2), m.group(3)) # d.init. tag-text tuple
        question_list = parse_questions(m.group(4))
        question_list.insert(0, node)
        case_list.append((tag, question_list))
    return(case_list)

# de Consecratione
def parse_part_3(text):
    distinction_list = []
```

```
    distinctions = re.findall('(?:\<1 DC\>)(.*?)(?=\<1 DC\>|$)', text)
    for distinction in distinctions:
        distinction = distinction.strip(' ')
        m = re.match('(\<2 \d\>) (\<4 1\>.*?)$', distinction)
        tag = m.group(1)
        canon_list = parse_canons(m.group(2))
        distinction_list.append((tag, canon_list))
    return(distinction_list)

def parse_questions(text):
    question_list = []
    questions = re.findall('(\<3 \d{1,2}\>.*?)(?=\<3 \d{1,2}\>|$)', text)
    for question in questions:
        question = question.strip(' ')
        m0 = re.match('(\<3 \d{1,2}\>) (\<T A\>) (.*?) (\<1 DP\>.*?)$', question) # C.33 q.3 (de Pen.)
        m1 = re.match('(\<3 \d{1,2}\>) (\<T A\>) (.*?) (\<4 1\>.*?)$', question)
        m2 = re.match('(\<3 \d{1,2}\>) (\<T A\>) (.*?)$', question) # C.11 q.2, C.17 q.3, C.22 q.3, C.29
q.1
        if m0:
            tag = m0.group(1)
            node = (m0.group(2), m0.group(3)) # d.a.c.1 tag-text tuple
            distinction_list = parse_de_pen(m0.group(4))
            question_list.append((tag, [node, ('<1 DP>', distinction_list)]))
        elif m1:
            tag = m1.group(1)
            node = (m1.group(2), m1.group(3)) # d.a.c.1 tag-text tuple
            canon_list = parse_canons(m1.group(4))
            canon_list.insert(0, node)
            question_list.append((tag, canon_list))
        elif m2:
            tag = m2.group(1)
            node = (m2.group(2), m2.group(3)) # d.a.c.1 tag-text tuple
            question_list.append((tag, [node]))
```

```
    return(question_list)

# de Penitentia
def parse_de_pen(text):
    distinction_list = []
    distinctions = re.findall('(?:\<1 DP\>)(.*?)(?=\<1 DP\>|$)', text)
    for distinction in distinctions:
        distinction = distinction.strip(' ')
        m = re.match('(\<2 \d\>) (\<T A\>) (.*?) (\<4 1\>.*?)$', distinction)
        tag = m.group(1)
        node = (m.group(2), m.group(3)) # d.a.c.1 tag-text tuple
        canon_list = parse_canons(m.group(4))
        canon_list.insert(0, node)
        distinction_list.append((tag, canon_list))
    return(distinction_list)

# return list of canons
def parse_canons(text):
    canon_list = []
    canons = re.findall('(\<4 \d{1,3}\>.*?)(?=\<4 \d{1,3}\>|$)', text)
    for canon in canons:
        canon = canon.strip(' ')
        m = re.match('(\<4 \d{1,3}\>) (.*?)$', canon)
        if m:
            nodes = parse_nodes(m.group(2))
        else: # C.1 q.4 c.6
            m = re.match('(\<4 \d{1,3}\>)$', canon)
            nodes = []
        canon_list.append((m.group(1), nodes))
    return(canon_list)

# return list of terminal nodes (tag-text tuples)
def parse_nodes(text):
```

```
    node_list = []
    nodes = re.findall('(\<T [AIPRT]\>.*?)(?=\<T [AIPRT]\>|$)', text)
    for node in nodes:
        node =node.strip(' ')
        m = re.match('(\<T [AIPRT]\>) (.*?)$', node)
        node_list.append((m.group(1), m.group(2)))
    return(node_list)

if __name__ == '__main__':
    main()
```