LING 331:
Text Processing for Linguists

Week 5

Basic Python cont. (More Assignment Notes)

More Notes from Assignment 2

- Counting ‘thy’s, problem?:
  - `grep 'thy' shakes.txt`
- Counting words, problem?:
  - `sed 's/ /\n/g' clean_shakes.txt | wc -l`
- Deleting blank lines, problem?:
  - `sed '/^$/d' shakes.txt`

More Notes from Assignment 2

- Getting ‘thy’ to ‘your’ lines, problem?

  -
    ```
    cat shakes.txt | sed s/thy/your/g | sed s/Thy/Your/g | grep "[Yy]our" | head
    ```
- Scripting (demo)

Notes from Assignment 3

(Feeding a fed horse)

PLEASE
MAKE
SURE
YOUR
ASSIGNMENT
RUNS!

`PYTHON ASSIGNMENT4.PY!!`

Notes from Assignment 3

- Periods at the end of a few of the initial `type`ing questions: … my bad.
    - E.g. `print('a. university + department.')`
- `g. expected_gpa and in_person`
    - How is this evaluated?
- `k. having_fun > total_students`
    - What's going on here?

Notes from Assignment 3

- `l. department[1532:]` vs. `m. department[1532]`
    - What’s going on here?

Notes from Assignment 3

- `for line in open(f)`
  Does not strip whitespace!
    - If you got 5-letter palindromes using min_length, this is because each line has ‘\n’ on the end!

Notes from Assignment 3

- `and` is not distributive
  - `type(d1) and type(d2) == int`
    is not the same as
    `type(d1) == int and type(d2) == int`
- The results of comparisons can be returned directly
  - E.g., no need for
    `if x == y return True else return False`
    Just do `return x == y`

Notes from Assignment 3

- `for` loops implicitly have a unit of operation:
    - For lists, `[‘abc’, 123, ‘you n me’]`
        - List item `‘abc’ -> 123 -> ‘you n me’`
    - For strings, `‘ling300’`
        - Character `‘l’ -> ‘i’ -> ‘n’ -> ‘g’`
    - For file objects, `open(f)`
        - Line

Notes from Assignment 3

- `in` gives useful functionality about presence/absence.
    - Is this char/string `in` this other one?
    - Is this item `in` this list or set or tuple?

Notes from Assignment 3

- `return` stops execution of a function.

```
if type(d1) != int or type(d2) != int is True:
    return not_dice_message
elif d1 > 6 or d1 < 1 or d2 > 6 or d2 < 1 is True:
    return not_dice_message
else: #[RV: Not necessary to have this else!]
    if d1 + d2 == 7 or d1 + d2 == 11 is True:
        return win_message
    else:
        return lose_message
```

Notes from Assignment 3

`==` vs. `is`

Logical equality — Object equality

```
>>> a = [1,2]
>>> b = [1,2]
>>> a == b   # are these logically equivalent?
True
>>> a is b   # are they the exact same object?
False
```

Notes from Assignment 3

- There’s a near-infinite variety of ways to do most things.

- Example: `reverse_string`
  - `s[::-1]`
  - `l = list(s), while len(l) > 0, l.pop()`
  - `l = list(s), l.reverse(), ' '.join(l)`
  - `i = len(s) - 1, while i > 0, i -= 1`
  - `new_s = '', for c in s, new_s = c + new_s`

Notes from Assignment 3

- Efficiency: not a huge deal for now, but be aware!
  e.g. consider how many times we loop over what

Which is better?

```
for word in s.split():     vs.  for word in stopwords:
    if word in stopwords:            if word in s.split():
```

- Anti-corollary: “Don’t optimize prematurely”
  Doing it whichever way is fine, until it gets too slow to work

Style Notes from Assignment 3

- Standards? Somewhat, e.g. style guide: https://www.python.org/dev/peps/pep-0008/
- Opinions? Many!

- Key consideration is **readability.**
    - Other people may have to read your code
    - You may have to read your own code in five years

Style Notes from Assignment 3

- Readability Basics:

    -
      ```
      # comments are good practice to explain the
      # purpose and functionality of more
      # complicated bits
      ```
    - The best code is also somewhat “self-documenting”
    - Variable names are a form of comment
    - Logical decomposition helps readability

Style Notes from Assignment 3

- Consider:

```
a = sum(vals)
b = len(vals)                vs.   return sum(vals)/len(vals)
return a/b
```

```
length1 = len(s1)
length2 = len(s2)          vs.  if len(s1) > len(s2):
if length1 > length2:                   ...
    ...
```

Style Notes from Assignment 3 (cont.)

- Variable naming: try not to overload (one name does one thing)

```
document = open(f)                     # file object
document = document.read()             # string
document = letters_only(document)      # string
document = document.split()            # list
```

vs.

```
document = open(f)                     # file object
text = letters_only(document.read())   # string
words = document.split()               # list
```

Style Notes from Assignment 3 (cont.)

- Nitpicky example from an assignment:

```
def reverse_string(s):
    rev_chars = ''
    count = len(s)   #[RV: My problem is here, why?]
    for char in s:
        count = count - 1
        let = s[count]
        rev_chars = rev_chars + let
    return rev_chars
```

Style Notes from Assignment 3 (cont.)

- Variable naming: try not to overload (one name does one thing)
  - Special case of this: `.join()`

🚫
```
output = ' '
output = output.join(words)
```

  - Both ‘`output`’s are strings, but they’re different - first is the delimiter, second is the actual output. Just do:

✔
```
output = ' '.join(words)
```

Style Notes from Assignment 3 (cont.)

- Simple is better and more readable. Try not to repeat yourself!

```
def vowel_count(s):
    index = 0
    low_s = s.lower()
    for l in low_s:
        if l == 'a':
            index = index + 1
        elif l == 'e':
            index = index + 1
        elif l == 'i':
            index = index + 1
```

```
...
        elif l == 'o':
            index = index + 1
        elif l == 'u':
            index = index + 1
        else:
            continue
    return index
```

Advanced Syntactic Sugar

- List Comprehension

```
output = ' '.join([c for c in s if c.isalpha()])
```

- Ternary Conditional Assignment

```
x = 0 if random.random() > 0.3 else 1
```

- Step slicing:

```
my_string[start:end:step]
```

Advanced Syntactic Sugar

- List Comprehension with Conditionals
  - nice example from an assignment!

```
def vowel_count(s):
    vowels = 'aeiouAEIOU'
    return sum(1 for c in s if c in vowels)
```

When You’re Stuck!

- `help(the_thing)`
- Read error messages carefully
- Carefully re-read the problem
- Talk your code out loud
- https://docs.python.org/3/
- Ed (try to explain the issue)
- Google it! (totally fine)
- Take a break
  (or skip the problem for now)
  and try again later

Demo

Let's talk through

`proportion_of_vowels_in_english`