Regex and Program Optimization
Code Snippets HW 2-9
A bit more about Regex
### Regex Tokens

<table>
<thead>
<tr>
<th>Token</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>Matches character exactly</td>
</tr>
<tr>
<td>[aeiouy]</td>
<td>Matches one of any character in the brackets</td>
</tr>
<tr>
<td>[a-z]</td>
<td>Matches a character in the range a-z</td>
</tr>
<tr>
<td>[a-zA-Z]</td>
<td>Matches a character in the range a-z or A-Z</td>
</tr>
<tr>
<td>\w</td>
<td>Matches any alphanumeric character and underscores</td>
</tr>
<tr>
<td>\s</td>
<td>Matches any whitespace (spaces, tabs, new lines)</td>
</tr>
<tr>
<td>\d</td>
<td>Matches any digit 0-9</td>
</tr>
<tr>
<td>.</td>
<td>Matches anything but a new line</td>
</tr>
</tbody>
</table>
# Regex Quantifiers

<table>
<thead>
<tr>
<th>Quantifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>If not specified, the token matches exactly one character</td>
</tr>
<tr>
<td>+</td>
<td>Matches 1 or more of preceding token</td>
</tr>
<tr>
<td>*</td>
<td>Matches 0 or more of preceding token</td>
</tr>
<tr>
<td>{2}</td>
<td>Matches a specific number of the preceding token</td>
</tr>
<tr>
<td>{1,3}</td>
<td>Matches a range from the lower up to and including the upper</td>
</tr>
<tr>
<td>{1,}</td>
<td>Matches the lower bound or more of the preceding token</td>
</tr>
<tr>
<td>{,5}</td>
<td>Matches up to and including the upper bound</td>
</tr>
</tbody>
</table>
Matching Start of String

Caret at the beginning of the pattern matches with the beginning of the string

```
pattern = '^\w+
matches = re.findall(pattern, text)
```

“Code didn't work, no idea why…
#overlyhonestmethods”
-mlkubik
Matching End of String

$ at end matches with the end of the string

pattern = '\w+$'
matches = re.findall(pattern, text)

“Code didn't work, no idea why…
#overlyhonestmethods”
-mlkubik
Matching End of String

Caret inside brackets inverts the range to exclude the provided tokens

```
pattern = '[^\w\s]+'
matches = re.findall(pattern, text)
```

“Code didn’t work, no idea why…
#overlyhonestmethods”

-mlkubik
Matching End of String

| symbol is like an or. The whole pattern matches if either expression matches

```python
pattern = '#\w+|@\w+
matches = re.findall(pattern, text)
```

"Code didn't work, no idea why…
`#overlyhonestmethods`  `@somebody"

-mlkubik
Fixing Slow Programs
Complexity of loops

- Loops are powerful tools for solving problems, but they also introduce complexity into our programs.

```python
count = 0
for i in range(100):
    count += 1
print(count)
```

What is count?
Complexity of loops

- Loops are powerful tools for solving problems, but they also introduce complexity into our programs.

```python
count = 0
for i in range(100):
    for j in range(100):
        count += 1

print(count)
```
Complexity of loops

• Loops are powerful tools for solving problems, but they also introduce complexity into our programs

```python
count = 0
for i in range(100):
    for j in range(100):
        for k in range(100):
            count += 1
print(count)
```

What is count?
Complexity of loops

- The number of times the most inner process is run is the product of the number of items iterated through for each loop.

```python
count = 0
for i in range(1000):
    for j in range(10):
        count += 1

print(count)
```
Complexity of loops

- With nested for loops, it doesn’t change the complexity if you change the order of the loop

```python
count = 0
for j in range(10):
    for i in range(1000):
        count += 1

print(count)
```

What is count?
Finding Single Occurrence Words

- How many times does this statement for the King James Bible occur?
- Number of words is 813,482
- 661,752,964,324

```python
words = text.split()
single_occur = 0
for word1 in words:
    word1_count = 0
    for word2 in words:
        if word1 == word2:
            word1_count += 1
    if count == 1:
        single_occur += 1
print(single_occur)
```
Finding Single Occurrence Words

• How many times does this statement for the King James Bible occur?
• Number of words is 813,482
• Vocabulary is 13149 words
• 28,067,940,322

words = text.split()
single_occur = 0
vocab = set(words)
for word1 in vocab:
    word1_count = 0
    for word2 in words:
        if word1 == word2:
            word1_count += 1
    if count == 1:
        single_occur += 1
print(single_occur)
Finding Single Occurrence Words

- How many times does this statement for the King James Bible occur?
- Number of words is 813,482
- Vocabulary is 13149 words
- 172,896,201
- ~291 seconds

```python
words = text.split()
single_occur = 0
count = 0
vocab = set(words)
for word1 in vocab:
    word1_count = 0
    for word2 in words:
        if word1 == word2:
            word1_count += 1
        if word1_count > 1:
            break
    if count == 1:
        single_occur += 1
    count = word1_count
print(single_occur)
```
Finding Single Occurrence Words

• Number of words is 813,482

• Vocabulary is 13149 words

• ~826,000 loop computations

• 0.15s

```python
words = text.split()
counter = {}
for word in words:
    if word not in counter:
        counter[word] = 0
    counter[word] += 1

single_occur = 0
for word, count in counter.items():
    if count == 1:
        single_occur += 1
```
Common Strategies for removing bottlenecks

• Iterate over as small of a collection as possible: will a set of the unique items do?

• Move as much as possible outside of the for loop

• Use a break to end a for loop early if possible

• Can you think of a Python command that will accomplish what your for loop accomplishes? Counter, sorted, reverse, max, min, sum, etc
Finding Bottlenecks

• Your program is taking a long time to complete, making debugging difficult

• What part of your code is your program getting stuck on?

```python
def main():
    result1 = my_function_a()
    result2 = my_function_b()
    result3 = my_function_c()
    print(result3)
```
Finding Bottlenecks

• Print intermediate results

• If it takes a long time to print out one of the intermediate results, you know you can look at the code that produced it

• By modularizing your code into functions, you can easily investigate the function themselves

```python
def main():
    result1 = my_function_a()
    print(result1)

    result2 = my_function_b()
    print(result2)

    result3 = my_function_c()
    print(result3)
```
Finding Bottlenecks

• Once you find a problematic function, look closely at it, does it contain a loop, or a nested loop?

• Can you reduce the number of items it loops through? (Iterate through a set/dictionary rather than a list)

• Can you precompute any steps outside of the for loop?

def my_function(words, themes):
    count = 0
    for word in words:
        themes_set = set(themes)
        if word in themes_set:
            count += 1
    return count
Finding Bottlenecks

• In for loops, the process step must be very fast for the entire for loop to be fast.

• Not all Python commands are equally fast.

```python
words = text.split()

num_short_words = 0
for word in words:
    if len(word) < 5:
        num_short_words += 1

num_sentences = 0
for word in words:
    num_sentences += word.count(' .')
```
Very Fast (ok for nested for loops)

- Operators +, -, *, /, ==, !=, <, >
- List indexing (words[i])
- Getting a value from a dictionary (counter[word])
- Converting a basic type to another (float(i), str(number))
- Adding an item to a list, set or dictionary
- Length of list, set, dictionary or strings (len(words))
- Checking if an item is in a set, dictionary or a string (word in vocab, ‘.’ in word)
Pretty fast, but dependent on size of collection (ok for single for loops, or shorter nested loops)

- Summarizing functions (max(), min(), sum())
- Converting a collection to another type of collection (set(words))
- Slicing of a list/string: words[:1000], text[1000:2000]
- Counting occurrences in a string
Not as fast (Ok for shorter for loops)
Very dependent on collection size

• Sorting a collection (sorted(words))

• Checking if an item is in a list (word in words), try to avoid. Can you create a set outside of the for loop?
Changes that won’t make any difference

• Removing comments

• Removing whitespace

• Making variable names or function names shorter

• Combining multiple lines into a single line

• Basically, don’t use bad coding practices in search of faster code. It won’t work
Timing code

• You identified that a complicated for loop which runs a critical piece of code takes too long, but you can’t reduce the number of elements it loops over without losing functionality

• Consider how you might change the steps inside the for loop to make them faster

• How can you measure how long it takes to run pieces of your code?

• Use Python’s built-in timing mechanisms to test which solutions are faster
time.time() Python’s wall clock

- time.time() tracks the number of seconds since the *epoch* (1/1/1970)
- Resolution of about 1 millisecond

```python
import time
current_time = time.time()
```
Using `time.time()` to time code

- Get the time at the start your code and at the end
- Print the elapsed time
- Reliable for elapsed times longer ~10 milliseconds

```python
import time

start = time.time()
result = my_function()
end = time.time()

diff = end - start
print("Elapsed: {}".format(diff))
```
Using time.time() to time code

- If you suspect a short routine inside a for loop is the culprit, time the short routine in a different for loop that runs for many iterations (1,000 - 100,000)

```python
def count_themes(words, themes):
    count = 0
    for word in vocab:
        if word in themes:
            count += 1
    return count
```
Using `time.time()` to time code

- Timing a short routine one time is unreliable.

- If you suspect a short routine inside a for loop is the culprit, time the short routine in a different for loop that runs for many iterations (1,000 - 100,000)

```python
start = time.time()
for i in range(10000):
    is_in = words[i] in themes

diff = time.time() - start
print("List: {}".format(diff))
```

```python
start = time.time()
themes = set(themes)
for i in range(10000):
    is_in = words[i] in themes

diff = time.time() - start
print("Set: {}".format(diff))
```