Week 09 Tutorial Sample Answers

1. Below are the current assignment autotests.

   Discuss what these print and why:

subset 0: quit

```
seq 42 44 | 2041 slippy 1q
```

ANSWER:

```
42
```

`1` is the address

`q` is the command

The addess `1` is the address of the first line.

The command `q` is the command to quit.

So `1q` will quit on the first line.

slippy will print the current line before it quits.

Giving us a single line of output: whatever the first line is.

In this case, the first line is `42`.

```
2041 slippy 10q < dictionary.txt
```

ANSWER:

```
...
...
the first 10 lines of the dictionary.txt file
```

`10` is the address

`q` is the command

The addess `10` is the address of the 10th line.

The command `q` is the command to quit.

So `10q` will quit on the 10th line.

slippy will print the current line before it quits.

Giving us 10 lines of output: the first 10 lines.

```
seq 41 43 | 2041 slippy 4q
```

ANSWER:

```
41
42
43
```

`4` is the address

`q` is the command

The addess `4` is the address of the 4th line.

The command `q` is the command to quit.

So `4q` will quit on the 4th line.

But as there are only 3 lines of input, slippy will hit EOF first.

Therefore, all three lines of output will be printed.

The `q` command never gets the chance to be used

The `q` command never gets the chance to be used.

```
seq 90 110 | 2041 slippy /.1/q
```

ANSWER:

```
90
91
```

`/.1/` is the address

`q` is the command

The addess `/.1/` is the address of any line that matches the regex `.1` .

The command `q` is the command to quit.

The line `91` matches the regex `.1` .

As the regex `.` (any character) matches the `9` .

So we will quit on the line `91` .

As allways print the current line before quitting.

```
2041 slippy '/r.*v/q' < dictionary.txt
```

ANSWER:

```
...
...
aardvark
```

`/r.*v/` is the address

`q` is the command

The addess `/r.*v/` is the address of any line that matches the regex `r.*v` .

The command `q` is the command to quit.

Depending on the contents of the dictionary.txt file, the lines printed may be different.

For my dictionary 247 line were printed before `aardvark` matches the regex `r.*v` .

All lines upto and including `aardvark` will be printed.

```
yes | 2041 slippy 3q
```

ANSWER:

```
y
y
y
```

Note: the `yes` command will print `y` infinitely.

`3` is the address

`q` is the command

The addess `3` is the address of the 3rd line.

The command `q` is the command to quit.

Because `yes` prints infinitely `slippy` can't wait untill EOF to stop.

`slippy` also can't read all input lines into an array.

`slippy` must process lines one at a time.

Note: because of the `$` address (last line) `slippy` needs to read two lines at a time.
The current lines and the next line (to detect when there is no next line).
`slippy` should not store more that two lines in memory at any time.

subset 0: print

```
seq 41 43 | 2041 slippy 2p
```

ANSWER:

```
41
42
42
43
```

Note: the `yes` command will print `y` infinitely.

`2` is the address

`p` is the command

The addess `2` is the address of the 2nd line.

The command `p` is the print command.

So `2p` will print on the second line.

This print is in addition to the automatic print of the current line that `slippy` already does.

This causes the second line to be printed twice.

```
head dictionary.txt | 2041 slippy 3p
```

ANSWER:

Third line of the dictionary.txt file is printed twice.

```
seq 41 43 | 2041 slippy -n 2p
```

ANSWER:

```
42
```

The `-n` option is used suppress (turn off) the automatic printing of the current line.

Therefore we only print when explicitly asked to.

We are asked to print the second line, and so get the output of `42` .

```
2041 slippy -n 42p < dictionary.txt
```

ANSWER:

Similar to the previous example

Only the 42nd line is printed.

```
head -n 1000 dictionary.txt | 2041 slippy -n '/z.$/p'
```

ANSWER:

Similar to the previous example

Only print a line if it matches the regex `z.$` .

That is: if the second last character is a `z` .

subset 0: substitute

```
seq 1 5 | 2041 slippy 's/[15]/zzz/'
```

ANSWER:

Run the substitute command on each line.

replace the *first* instance of `1` or `5` on each line with `zzz`.

```
seq 1 5 | 2041 slippy 's/[15]/zzz/g'
```

ANSWER:

Run the substitute command on each line.

replace *all* instances of `1` or `5` with `zzz`.

```
echo "Hello Andrew" | 2041 slippy 's/e//'
```

ANSWER:

Run the substitute command on each line.

replace the *first* instance of `e` on each line with the empty string.

```
echo "Hello Andrew" | 2041 slippy 's/e//g'
```

ANSWER:

Run the substitute command on each line.

replace *all* instances of `e` with the empty string.

subset 1: addresses

```
seq 1 5 | 2041 slippy '$d'
```

ANSWER:

`$` is the special address for the last line.

`d` is the delete command.

If a line is deleted then processing immediately moves on to the next line.

The line is not automatically printed.

```
seq 42 44 | 2041 slippy 2,3d
```

ANSWER:

`2,3` is a range address.

The command is applied to all lines within the range (start and end line inclusive).

```
seq 10 21 | 2041 slippy 3,/2/d
```

ANSWER:

Similar to the previous example

```
seq 10 21 | 2041 slippy /2/,7d
```

ANSWER:

Similar to the previous example

```
seq 10 21 | 2041 slippy /2/,/7/d
```

ANSWER:

Similar to the previous example

subset 1: substitute

```
seq 1 5 | 2041 slippy 'sX[15]XzzzX'
```

subset 1: multiple commands

```
seq 1 5 | 2041 slippy '4q;/2/d'
```

subset 1: -f

```
echo "4q" > commands.script
echo "/2/d" >> commands.script
seq 1 5 | 2041 slippy -f commands.script
```

subset 1: input files

```
seq 1 2 > two.txt
seq 1 5 > five.txt
2041 slippy '4q;/2/d' two.txt five.txt
```

subset 1: whitespace

```
seq 24 42 | 2041 slippy ' 3, 17  d  # comment'
```

subset 2: -i

```
seq 1 5 > five.txt
2041 slippy -i /[24]/d five.txt
cat five.txt
```

subset 2: multiple commands

```
echo 'Punctuation characters include . , ; :' | 2041 slippy 's/;/semicolon/g;/;/q'
```

2. Write a Python program, `tags.py` which given the URL of a web page fetches it by running *wget* and prints the HTML tags it uses.

   The tag should be converted to lower case and printed in alphabetical order with a count of how often each is used.

   Don't count closing tags.

   Make sure you don't print tags within HTML comments.

```
$ ./tags.py https://www.cse.unsw.edu.au
a 141
body 1
br 14
div 161
em 3
footer 1
form 1
h2 2
h4 3
h5 3
head 1
header 1
hr 3
html 1
img 12
input 5
li 99
link 3
meta 4
noscript 1
p 18
script 14
small 3
span 3
strong 4
title 1
ul 25
```

   Note the counts in the above example will not be current - the CSE pages change almost daily.

ANSWER:

```
#! /usr/bin/env python3

# written by Nasser Malibari and Dylan Brotherston
# fetch specified web page and count the HTML tags in them

import sys, re, subprocess
from collections import Counter

def main():

    if len(sys.argv) != 2:
        print(f"Usage: {sys.argv[0]} <url>", file=sys.stderr)
        sys.exit(1)

    url = sys.argv[1]

    process = subprocess.run(["wget", "-q", "-O-", url], capture_output=True, text=True)
    webpage = process.stdout.lower()

    # remove comments
    webpage = re.sub(r"<!--.*?-->", "", webpage, flags=re.DOTALL)

    # get all tags
    # note: use of capturing in re.findall returns list of the captured part
    tags = re.findall(r"<\s*(\w+)", webpage)

    # using collections.counter, alternatively can use a dict to count
    tags_counter = Counter()
    for tag in tags:
        tags_counter[tag] += 1

    for tag, counter in sorted(tags_counter.items()):
        print(f"{tag} {counter}")

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

3. Add an `-f` option to `tags.py` which indicates the tags are to be printed in order of frequency.

```
$ ./tags.py -f https://www.cse.unsw.edu.au
head 1
noscript 1
html 1
form 1
title 1
footer 1
header 1
body 1
h2 2
hr 3
h4 3
span 3
link 3
small 3
h5 3
em 3
meta 4
strong 4
input 5
img 12
br 14
script 14
p 18
ul 25
li 99
a 141
div 161
```

ANSWER:

```python
#! /usr/bin/env python3

# written by Nasser Malibari and Dylan Brotherston
# fetch specified web page and count the HTML tags in them

import re, subprocess
from collections import Counter
from argparse import ArgumentParser

def main():

    parser = ArgumentParser()
    parser.add_argument('-f', '--frequency', action='store_true', help='print tags by
frequency')
    parser.add_argument("url", help="url to fetch")
    args = parser.parse_args()

    process = subprocess.run(["wget", "-q", "-O-", args.url], capture_output=True, text=True)
    webpage = process.stdout.lower()

    # remove comments
    webpage = re.sub(r"<!--.*?-->", "", webpage, flags=re.DOTALL)

    # get all tags
    # note: use of capturing in re.findall returns list of the captured part
    tags = re.findall(r"<\s*(\w+)", webpage)

    # using collections.counter, alternatively can use a dict to count
    tags_counter = Counter()
    for tag in tags:
        tags_counter[tag] += 1

    if args.frequency:
        for tag, counter in reversed(tags_counter.most_common()):
            print(f"{tag} {counter}")
    else:
        for tag, counter in sorted(tags_counter.items()):
            print(f"{tag} {counter}")

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

4. Modify tags.py to use the `requests` and `beautifulsoup4` modules.

ANSWER:

```
#! /usr/bin/env python3

# written by Dylan Brotherston
# fetch specified web page and count the HTML tags in them

from collections import Counter
from argparse import ArgumentParser

import requests
from bs4 import BeautifulSoup

def main():

    parser = ArgumentParser()
    parser.add_argument('-f', '--frequency', action='store_true', help='print tags by
frequency')
    parser.add_argument("url", help="url to fetch")
    args = parser.parse_args()

    response = requests.get(args.url)
    webpage = response.text.lower()

    soup = BeautifulSoup(webpage, 'html5lib')

    tags = soup.find_all()
    names = [tag.name for tag in tags]

    tags_counter = Counter()
    for tag in names:
        tags_counter[tag] += 1

    if args.frequency:
        for tag, counter in reversed(tags_counter.most_common()):
            print(f"{tag} {counter}")
    else:
        for tag, counter in sorted(tags_counter.items()):
            print(f"{tag} {counter}")

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

5. If you fell like a harder challenge after finishing the challenge activity in the lab this week have a look at the following websites for some problems to solve using regexp:
   - https://regex101.com/quiz
   - https://alf.nu/RegexGolf