CS 35L Week 3 Worksheet

Emacs Lisp & Python Scripting Worksheet

Emacs Lisp:

The following questions are to practice understanding Emacs Lisp. A helpful resource is using this cheat sheet.

Syntax!

Recall that Lisp is different from other languages – Lisp uses `(fxn a b)`

1. What do the following Lisp expressions output, when evaluated?
   a. `(quote (1 2 3))`
      `(1 2 3)`
   b. `'(1 2 3)`
      `(1 2 3)`
   c. `(list (+ 1 2) '(+ 1 2))`
      `(3 (+ 1 2))`
   d. `(cons (+ 1 2) '(3 4))`
      `(3 3 4)`
   e. `(+ 10 (car '(1 2 3)))`
      `11`
   f. `(reverse (append '(1 2) '(3 4)))`
      `(4 3 2 1)`
   g. `(cdddar '(1 2 3 4 5 6 7))`
      This is illegal! `car` gives 1st item of a pair. `cdr` gives tail. `(cddr x)` is the same as `(cdr (cdr x))`. `cdddar` would correspond to `(cdr (cdr (cdr (car x))))` – however, `x` in this case is a single list – `car` would result in a single entry, so `cdr` of `car` of `x` would result in an error, since the return types don't match up. However, `cadddr` WOULD work. It would return 4
   h. `Convert the C expression (3 + 2 * 5 > 0 && 7 < 3 / 5) to equivalent Emacs Lisp`
      `(and (> (+ 3 (* 2 6)) 0) (< 7 (/ 3 5)))`

2. Dr. Eggert mentioned in lecture that you can edit your Emacs source code to customize it to your liking. One source of confusion is how the indenting works with tabs. Can you adjust your Emacs settings to set tab indenting to 4 spaces?

   (setq-default indent-tabs-mode nil)

(setq-default tab-width 4)
(setq indent-line-function 'insert-tab)

a. Where are the configuration files located for Emacs?
   Check for init.el, possible inside ~/.emacs.d
b. How do we create a configuration file if none exist?
   Touch, emacs, and other commands can create files.

1. Write a function called is-current-line-even that takes no arguments, and uses the message function to print out whether the current line number is even or odd.
   a. Example: If the current line is 11, (is-current-line-even) should print "even" to the echo area.
   b. Example: If the current line is 4, (is-current-line-even) should print "odd" to the echo area.
   c. Add the interactive keyword to make your function callable from Emacs.

```
(defun is-current-line-even ()
  "Print whether the current line number is even or odd."
  (interactive "Print out whether the current line number is even or odd")
  (setq x (line-number-at-pos))
    (if (= (% x 2) 0)
        (message "even")
      (message "odd")))

; Without using line-number-at-pos and count-lines instead
(defun is-current-line-even ()
  "Print whether the current line number is even or odd using count-lines."
  (interactive)
  (save-restriction
    (widen)
    (save-excursion
      (beginning-of-line)
      (let ((x (1 + (count-lines 1 (point))))
        (if (= (% x 2) 0)
            (message "even")
          (message "odd"))))))
```

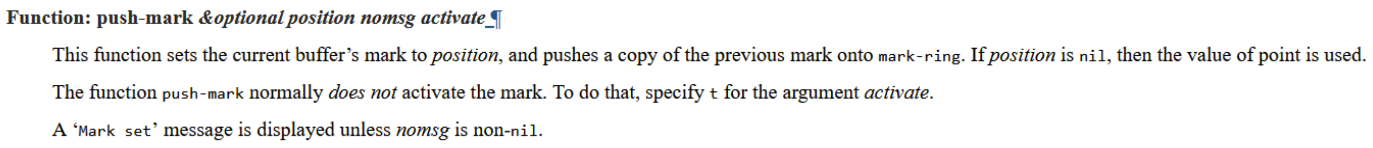
https://www.math.utah.edu/docs/info/emacs-lisp-intro_7.html#:~:text=The%20save%2Drestriction%20special%20form%20is%20followed%20by%20widen%20.,that%20save%2Drestriction%20remembers.)

Week 3: Emacs Lisp & Python Scripting Worksheet

1. Let’s look at a Lisp function that acts on buffers, mark-whole-buffer.

```
1   (defun mark-whole-buffer ()
2     "Put point at beginning and mark at end of buffer.
3   You probably should not use this function in Lisp programs;
4   it is usually a mistake for a Lisp function to use any subroutine
5   that uses or sets the mark."
6     (interactive)
7     (push-mark (point))
8     (push-mark (point-max) nil t)
9     (goto-char (point-min)))
```

   a. Look up the push-mark documentation. What is happening in line 8?



Function: push-mark ***&optional position nomsg activate*** ¶

This function sets the current buffer’s mark to *position*, and pushes a copy of the previous mark onto `mark-ring`. If *position* is `nil`, then the value of point is used.

The function `push-mark` normally *does not* activate the mark. To do that, specify `t` for the argument *activate*.

A ‘`Mark set`’ message is displayed unless *nomsg* is non-`nil`.

The push-mark documentation is above. In line 8, as we’ve specified t for the activate argument, we’re activating the mark (point-max) with no ‘Mark set’ message.

   b. With this in mind, explain what each line of the 3 lines of the mark-whole-buffer body (lines 7-9) do.

Line 7 pushes the mark where the cursor usually is. This is for the convenience of the user. Line 8 activates the mark of point-max, so our mark is active at the very end of the buffer. Finally, line 9 puts us at point-min, the very beginning of the buffer, and as a result we have marked the entire buffer.

2. Let's look at another Lisp function, which we will call **foo**.

```
1  (defun foo (buffer start end)
2    (interactive "bExplanatory text here: \nr")
3    (let ((oldbuf (current-buffer)))
4      (with-current-buffer (get-buffer-create buffer)
5        (barf-if-buffer-read-only)
6        (erase-buffer)
7        (save-excursion
8          (insert-buffer-substring oldbuf start end)))))
```

a. In line 4, `(get-buffer-create buffer)` tries to get `buffer`; if it can't, it creates it.

   i. With this in mind, and noting that the scope of `with-current-buffer` extends *past* line 4, what do you think `with-current-buffer` does?

   with-current-buffer is a function that evaluates its body with the buffer passed to it.

   ii. Why would this be useful?

   This is essentially a cleaner way to do save-excursion and set-buffer.

b. What might `insert-buffer-substring` do? (Hint: We know that oldbuf is bound to the user's current buffer, and we see that we are also giving insert-buffer-substring start and end)

   insert-buffer-substring copies the text into the current buffer from the region indicated by start and end in oldbuf.

c. With a and b in mind, what does foo do, and what might "Explanatory text here" actually say?

   The r in the last line of interactive denotes two arguments: the point and mark, with the smaller one going first. Thus, they are the start and end of the current selection. This means that foo will replace the contents of a specified buffer with the selected text in the current buffer. "Explanatory text here" might say something like "bCopy to buffer: " since it takes a buffer argument.

3. Now let's look at a second Lisp function, which we will call **bar**.

```
1   (defun bar (buffer)
2     "Explanatory text here.
3     BUFFER may be a buffer or a buffer name."
4     (interactive "BMore explanatory text: ")
5
6     (or (bufferp buffer)
7         (setq buffer (get-buffer buffer)))
8     (let (start end newmark)
9       (save-excursion
10        (save-excursion
11          (set-buffer buffer)
12          (setq start (point-min) end (point-max)))
13
14        (insert-buffer-substring buffer start end)
15        (setq newmark (point)))
16      (push-mark newmark)))
```

   a. The tricky part of determining what bar does is `bufferp`. As a hint, `bufferp` is a predicate that tells us if buffer is a buffer or not. With this in mind, what do lines 6 and 7 accomplish?

      In lines 6 and 7, we check if buffer is a buffer or the name of a buffer. If it's a buffer, we use the buffer, and if it is just the buffer name, we get the buffer using get-buffer. Thus, after lines 6 and 7 we know that buffer must be the buffer we want (rather than just its name).

   b. Now, let's look at the *inner* `save-excursion` first.

```
(save-excursion
  (set-buffer buffer)
  (setq start (point-min) end (point-max)))
```

      i. What does the set-buffer accomplish?

         Changes to the argument buffer.

      ii. What does the setq accomplish?

         We set the variable start to point-min and the variable end to point-max.

iii. What will happen once this inner save-excursion is executed? I.e., what will remain true in the *outer* save-excursion?

start and end will remain the point-min and point-max of the argument buffer (NOT the user's current buffer).

c. Now we look at the outer save-excursion.

```
(save-excursion                                   1
  (inner-save-excursion)                          2
  (insert-buffer-substring buffer start end)      3
  (setq newmark (point)))                         4
```

i. What happens in line 3?

In line 3, we insert all of the contents of our argument buffer into our current buffer using the argument buffer's point-min and point-max.

ii. What point is being saved to newmark, and why might we later want to push this newmark?

We're saving the end of the copied content. We might want to push this later because when you're copying and pasting text into a buffer it's a common workflow to then want to directly act on that text.

d. With a, b, and c in mind, what does bar do, and what might "Explanatory text here" actually say?

bar copies the contents of the argument buffer into your current buffer.

"Explanatory text here" might say something like "Insert after point the contents of BUFFER. Puts mark after the inserted text."

4. Let’s say we now want to write our own function in Lisp, and it will do something buffer-related which we find helpful. We’ll call the function buffer-op.
   a. Define buffer-op with no body and taking no arguments.
   ```
   (defun buffer-op ())
   ```
   b. Add a line to part a which will allow buffer-op to be invoked by a user.
   ```
   (defun buffer-op ()
               (interactive))
   ```
   c. What do we need to add if buffer-op has to take a buffer argument?
   ```
   (defun buffer-op (buffer)
               (interactive “bBuffer for operation: “))
   ```
   d. Using (body) as a placeholder for the actual function body, add a line to part b which will allow us to ensure that the user’s cursor doesn’t change as a result of buffer-op.
   ```
   (defun buffer-op (buffer)
               (interactive “bBuffer for operation: “)
               (save-excursion
                           (body))))
   ```

5. **(S22 Midterm)** Fix three bugs in the following Emacs Lisp function, which is to be executed in View mode when you type ‘%’. The bugs cause the code to not agree with its documentation.

```
(defun View-goto-percent (&optional percent)
  "Move to end (or prefix PERCENT) of buffer.
Center display at point.
Also set the mark at the position where point was."
  (interactive "P")
  (push-mark)
  (view-recenter)
  (goto-char
    (if percent
      (+ (point-min)
          (floor
              (* (- (point-max) (point-min))
            (max 0 (min 100 (prefix-numeric-value percent))))))
      (point-min))))
```

Solution:

- Bug 1: view-recenter should be after goto-char
- Bug 2: percent should be divided by 100
- Bug 3: the default value in the last line should be point-max

Python Scripting

The following questions are to practice understanding Python. A helpful resource is the Python docs for sys, random, sorting, and argparse.

1. Write a Python function that:
   a. Takes a list of lists as its only parameter,
   b. Randomly shuffles the items in each inner list,
   c. Then, randomly shuffles the order of the elements of the main outer list.

Solution:

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

def shuf(x):
     for list_i in x:
          random.shuffle(list_i)
     random.shuffle(x)
```

2. ArgParse Activity

Write a Python script that takes in a range of numbers, generates a random number within that range, and prompts the user for input. If the user guesses the number correctly, they win. Otherwise, they lose. You should use the *random* and *argparse* libraries..

a. "-r, --range" which takes in numbers in the form n-m and generates a number (inclusive) in the range
b. "-t, --tries" takes a number and fails the user if they guess too much

Examples:

```
$python guess.py -r 4-6 -t 1
 >5
Wrong you fail :(

$python guess.py -r 6-6
>2
Nope
>6
Yay you win :)

$python guess.py -r 4-3
exited with error 1: you cannot have a negative range :(
```

Solution:

```
#!/usr/bin/env python3
import argparse
import random
import sys

# Setup argument parser
parser = argparse.ArgumentParser(description="A number guessing
game")
parser.add_argument("-r", "--range", type=str, required=True,
help="Range in the format n-m (inclusive)")
parser.add_argument("-t", "--tries", type=int, default=1,
help="Number of tries (default is 1)")

# Parse the command-line arguments
args = parser.parse_args()

# Extract and validate the range argument
try:
    range_values = args.range.split("-")
    if len(range_values) != 2:
        raise ValueError("Invalid format for range. Use n-m.")
    start, end = int(range_values[0]), int(range_values[1])

    if start > end:
        sys.stderr.write("Exited with error 1: you cannot have a
negative range :(\n")
        sys.exit(1)
except ValueError as e:
    sys.stderr.write(f"Exited with error 1: {e}\n")
    sys.exit(1)

# Generate the random number to guess
number_to_guess = random.randint(start, end)

# Set the number of tries
tries = args.tries

# Main game loop
while tries > 0:
    try:
        guess = int(input("> "))
    except ValueError:
        sys.stdout.write("Invalid input, please enter a number.\n")
        continue

    if guess == number_to_guess:
        sys.stdout.write("Yay you win :)\n")
        break
    else:
        tries -= 1
        if tries > 0:
            sys.stdout.write("Wrong! Try again.\n")
        else:
            sys.stdout.write("Wrong you fail :(\n")
```

3. **File System Activity**

Write a Python script that, when given a path, i.e., “*/my/path*”, will read the file's content and print the first word on each line. To do this, you’ll need to use sys.

For example:

```
$cat ~/myfile.txt
Hi im a file
I like to store things
Please don't delete me or I will die
My life is in your hands :)

$python3 mycat.py myfile.txt
Hi
I
Please
My
```

Solution:

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

if len(sys.argv) != 2:
     sys.stderr.write("Missing file name\n")
     sys.exit(1)

filename = sys.argv[1]
with open(filename, "r") as f:
     lines = f.readlines()

for line in lines:
     first_word = line.strip().split(" ")[0]
     print(first_word)
```

4. **File System Activity**

Write a Python script that, in a random order, prints out all names of files and directories in a specified directory (you **don't need to worry about hidden files** that start with a **'.'**, such as **.bashrc**). You'll need to use *sys* to read in the path and *os.listdir()* to get all file names.

Example:

```
$ls /my/path
me.txt
Hi
Acm

$python randls.py /my/path
Acm
me.txt
Hi
```

Solution:

```python
#!/usr/bin/env python3
import os, random, sys

if len(sys.argv) != 2:
     sys.stderr.write("Missing file path\n")
     sys.exit(1)

path = sys.argv[1]
files = os.listdir(path)
random.shuffle(files)

for file in files:
     print(file)
```

5. Argparse activity

Write a Python implementation of the bash command *echo*. In case you aren't familiar with it, *echo* reads in standard input and prints what was read in. *echo* has a few flags you need to implement and a few custom ones.

- "-n" says do not output a trailing new line. You might find this useful
- "--help" which prints out the help page for all the flags. You might find this helpful
- "-c" which adds a smiley face :) after everything it prints (not in the bash version but I think its fun)
- "-r, --repeat" which prints out the echo'd value as many times as the number following r
- "-f, –file" which takes in a file name and echos the contents (basically cat)

Here are some examples:

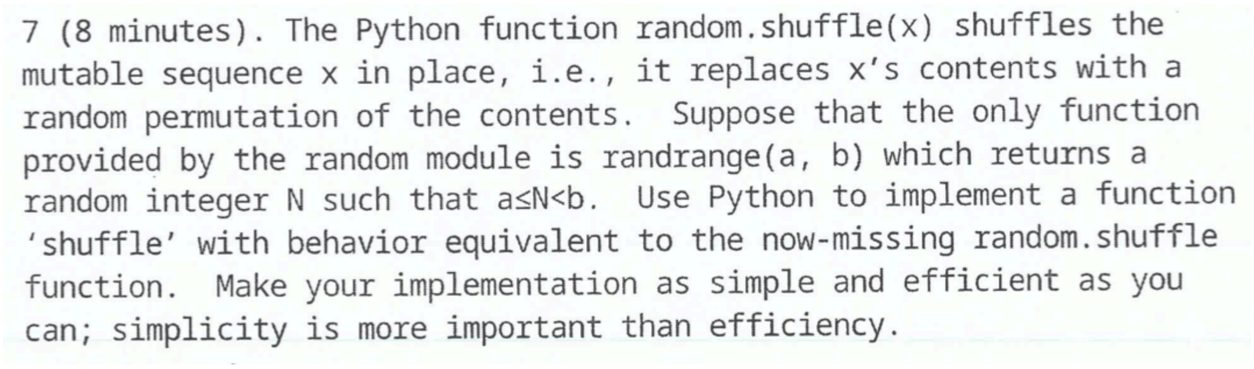
```
$echo -r 2 -c hi
hi :)
hi :)

$echo -n -r 3 benson
benson
benson
benson$command2 #this isn't an output but just to show that there is no
training new line

$echo -f hi.txt
hi i'm trapped in a file factory, whenever you type mkdir or touch they
make me go onto your disk or ssd and make a new file :(
```

Solution:

```
#!/usr/bin/env python3
import argparse, sys

parser = argparse.ArgumentParser("weirdecho")

parser.add_arguments("ARGS", nargs="*")
parser.add_argument("-n", "--no-newlines", action="store_true")
parser.add_argument("-c", "--character", action="store_true")
parser.add_argument("-r", "--repeat", type=int)
parser.add_argument("-f", "--file", type=str)
```

```python
args = vars(parser.parse_args())

args_option = args.get("ARGS")
no_newlines_option = args.get("no_newlines")
character_option = args.get("character")
repeat_option = args.get("repeat")
file_option = args.get("file")

contents = []

if file_option:
     with open(file_option, "r") as f:
         contents = f.readlines()
else:
     contents = args_option

def print_lines():
     end_character = ""
     if character_option:
         end_character += " :)"
    if not no_newlines_option:
         end_character += "\n"
   for line in content:
         print(line, end=end_character)

if repeat_option:
     for _ in range(repeat_option):
          print_lines()
else:
     print_lines()
```

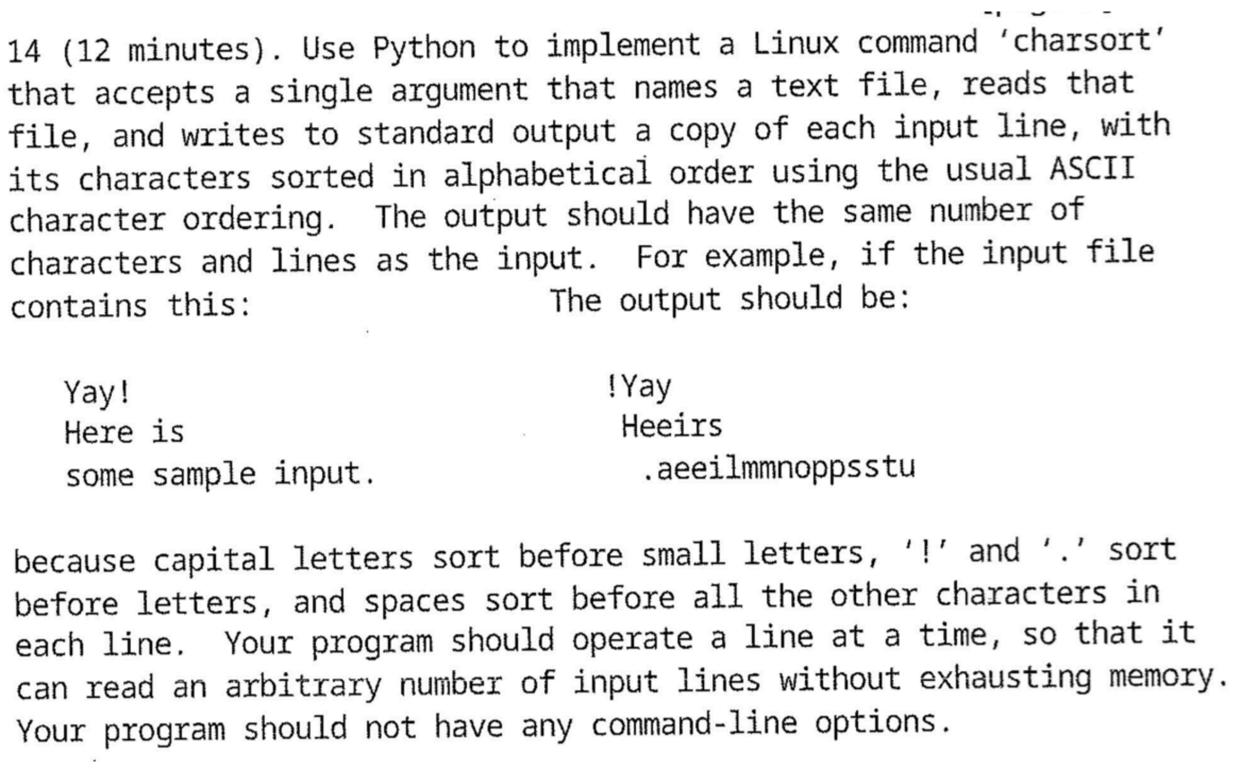
6. **CS 35L Spring 2023 Midterm**. This demonstrates a common CS 35L exam question where you must write a Python function. A frequent theme involves taking some part of one of the assignments and implementing it slightly differently.



7 (8 minutes). The Python function random.shuffle(x) shuffles the mutable sequence x in place, i.e., it replaces x's contents with a random permutation of the contents. Suppose that the only function provided by the random module is randrange(a, b) which returns a random integer N such that a≤N<b. Use Python to implement a function 'shuffle' with behavior equivalent to the now-missing random.shuffle function. Make your implementation as simple and efficient as you can; simplicity is more important than efficiency.

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

def shuffle(x):
     for i in range(len(x)):
         j = random.randrange(i, len(x))
         x[i], x[j] = x[j], x[i]
```

7. **CS 35L Fall 2023 Midterm.** This demonstrates a common CS 35L exam question where you must implement a simple script that tackles some problems. Common themes involve list and string manipulation and argument handling with **sys** and **argparse**.



14 (12 minutes). Use Python to implement a Linux command 'charsort' that accepts a single argument that names a text file, reads that file, and writes to standard output a copy of each input line, with its characters sorted in alphabetical order using the usual ASCII character ordering. The output should have the same number of characters and lines as the input. For example, if the input file contains this: The output should be:

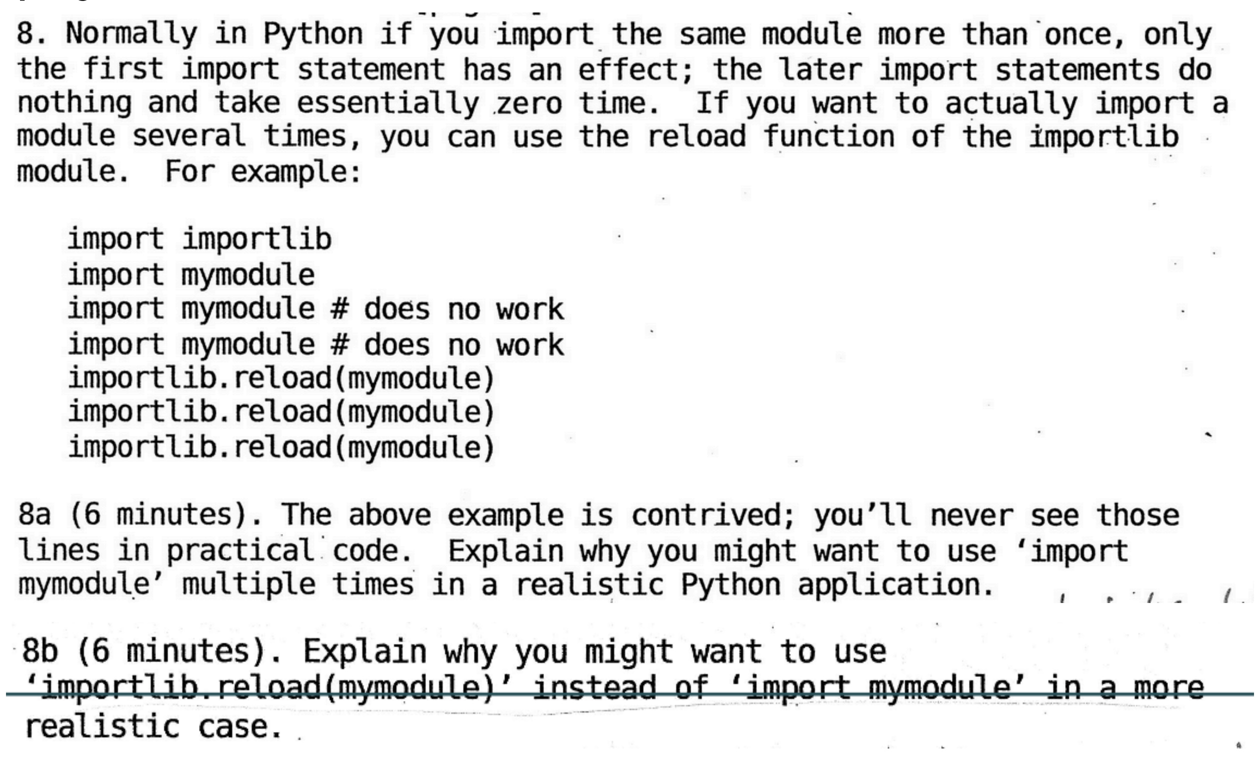
```
Yay!                          !Yay
Here is                        Heeirs
some sample input.               .aeeilmmnoppsstu
```

because capital letters sort before small letters, '!' and '.' sort before letters, and spaces sort before all the other characters in each line. Your program should operate a line at a time, so that it can read an arbitrary number of input lines without exhausting memory. Your program should not have any command-line options.

```python
#!/usr/bin/env python3
import argparse, sys

def main():
    parser = argparse.ArgumentParser(add_help=False)
    parser.add_argument('file')

    args = parser.parse_args(sys.argv[1:])

    with open(args.file) as f:
        for line in f:
            sorted_chrs = ''.join(sorted(line.strip()))
            print(sorted_chrs)

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

Python Modules

Spring 2022 Midterm



8. Normally in Python if you import the same module more than once, only the first import statement has an effect; the later import statements do nothing and take essentially zero time. If you want to actually import a module several times, you can use the reload function of the importlib module. For example:

```
import importlib
import mymodule
import mymodule # does no work
import mymodule # does no work
importlib.reload(mymodule)
importlib.reload(mymodule)
importlib.reload(mymodule)
```

8a (6 minutes). The above example is contrived; you'll never see those lines in practical code. Explain why you might want to use 'import mymodule' multiple times in a realistic Python application.

8b (6 minutes). Explain why you might want to use 'importlib.reload(mymodule)' instead of 'import mymodule' in a more realistic case.

8a.

- You use mymodule in multiple files within the Python application, thus it is imported multiple times
- Readability
- Using in conditional statements

8b.

- Most used with long-running interactive sessions like Jupyter notebooks where you update a file and want to re-import it without restarting the whole notebook
- Want to load the module (corrected version) again without restarting the whole code from scratch. (Example: While debugging)
- Done when in interactive mode