CS 35L
Discussion 1A
Week 4

Midterm Review

Contents

1. Logistic

- In-person Midterm (Next Wednesday 2PM, Oct 26, in class, Haines Hall 39)
- Timing
  - 1 min ⇔ 1pts
  - Please arrive on time to make sure you can fully make use of the exam minutes
- Resource
  - Test is open notes, open books, open any paper-based materials
  - **NO** Internet
  - **NO** computer / phone
  - **NO** collaboration / communication with others
- What if I cannot attend in time?
  - Please contact professor Eggert if you can not make it
  - He will probably give other forms of midterms

Contents

2. Strategy

- Make use of the time you have!
  - Check how many points (minutes) for each questions
    - Usually Qs with fewer points are okay to have shorter answers
    - Usually Qs with larger points will take longer time
  - Easy ones first:
    - First answer as many questions as you easily can
    - Then come back later to the hard ones
  - Don’t worry about the perfect answers:
    - Try to move on and get as many points as you can
- For hard questions:
  - Try to focus more on the **overall approach** for more points, then come back for small details (which might be worth fewer points)
  - If Qs ask you to **illustrate with an example**, try to give a **specific example**
  - There could be many correct answers:
    - If Qs ask you to explain, try to focus more on concrete justification for your answer (your example)

2. Strategy

- Review your HWs/Labs
- Make use of the resources (personal suggestions)
  - Print out / summarize some resources
    - Summaries: basic **concepts** Prof. mentioned in his lecture
    - **Simple** examples for Pipe / IO redirection operators
    - Summarize some **common** linux shell commands
    - Regular Expression patterns
    - (Most)Basic Syntax for shell scripting
    - Emacs reference card
    - (Most) Basic Syntax for Lisp
    - (Most) Basic Syntax for python
    - (Most) Basic Syntax for HTML
    - (Most) Basic Syntax for Javascript
  - **Basic concepts and ideas** are more important than concrete syntax for paper-based exams
  - Don't bring too many different kinds of resources; don't bring heavy books (it will distract you; make sure you can locate what you need in a fast manner)
  - **Print and bring all HWs with you (including your code)**

Contents

3. Overview (from Syllabus)

- Files, Shell and shell scripting
  - OS / Process / Unix file system / POSIX
  - Basic shell commands: man, ls, pwd, cd, mkdir, rmdir, echo, cat, cp, mv, ln, rm, chmod, kill, ps, grep, find
  - Pattern matching, Regular Expression (Basic vs Extended)
  - Pipeline and in/out redirection a < b | c > d
  - Scripting
- Files, editing
  - Emacs and basic commands
  - Emacs Lisp: Define and using functions
- Python
  - Variables, control flow, functions, classes, modules .....
  - Virtual Environment (if Prof. introduce this idea in his lecture) - Package Management
- Client-server apps and user interfaces
  - HTML / JavaScript / React

3. Practice - Sample Midterm - 1

1 (2 minutes). What is the likely typo in the following shell command, and why is it such a serious typo?

```
rm *.[Oo][Uu][Tt] * .o *.a
```

3. Practice - Sample Midterm - 1

```
1 (2 minutes).  What is the likely typo in the following shell
command, and why is it such a serious typo?

  rm *.[Oo][Uu][Tt] * .o *.a
```

- What does command `rm` do?
- What does `*` means in the shell context?
- What if I do `rm *`

3. Practice - Sample Midterm - 1

```
1 (2 minutes).  What is the likely typo in the following shell
command, and why is it such a serious typo?

  rm *.[Oo][Uu][Tt] * .o *.a
```

- What does command `rm` do?
- What does `*` means here?
- What if I do `rm *`

- Extra space in between `*` and `.o`
- Will delete everything in the directory instead of deleting files ended with `.o`

3. Practice - Sample Midterm - 2

```
2. A set of read/write/execute permissions on a file is called
“sensible” if the owner has all the permissions of the group, and the
group has all the permissions of others.  For example, 551 (octal) is
sensible, whereas 467 (octal) is not.

2a (2 minutes). Briefly explain why non-sensible permissions don’t
make much sense.
```

- This is typical question in midterms/finals
  - Introduce a new idea/concept in the description
  - Then several follow up questions

- **Understand the new definition first**!
  - Every follow up question will make use of this concept
  - Combine with what you have already learned

3. Practice - Sample Midterm - 2

```
2. A set of read/write/execute permissions on a file is called
“sensible” if the owner has all the permissions of the group, and the
group has all the permissions of others.  For example, 551 (octal) is
sensible, whereas 467 (octal) is not.

2a (2 minutes). Briefly explain why non-sensible permissions don’t
make much sense.
```

- **Understand the new definition/concept first**!
  - Every follow up question will make use of this concept
  - Combine with what you have already learned

- What is “sensible” permission?
- Permission: Owner **?** Groups **?** Others
- Who’s the owner? Who’s the groups? Who’s the others?

3. Practice - Sample Midterm - 2a

```
2. A set of read/write/execute permissions on a file is called
“sensible” if the owner has all the permissions of the group, and the
group has all the permissions of others.  For example, 551 (octal) is
sensible, whereas 467 (octal) is not.

2a (2 minutes). Briefly explain why non-sensible permissions don’t
make much sense.
```

- Understand the definition of “sensible” permissions first!
  - Permission: Owner >= Groups >= Others
  - Who’s the owner? Who’s the groups? Who’s the others?

- Owner: the person who is considered the primary owner of the file (the closest relation to the file)
- Group: a group of people who owns(connected to) the file in some way (but not as much as the owner) (the next closest relation to the file)
- Other: all users other than owner and group (the least relation to the file)
- People with less relation to the file should NOT have more permissions

3. Practice - Sample Midterm - 2b

```
2b (6 minutes). How many distinct sensible permissions are there?
Explain.
```

- Understand the definition of “sensible” permissions first!
  - Permission: Owner >= Groups >= Others

1. List all possibilities and count

If other’s bits all 0: 000 (1 case)				
Possible Group bits	000	100, 010, 001	110, 011, 101	111
#Possible owner bits	All 8 cases	4 per case	2 per case	1
#sub-total	8	3x4 = 12	3x2 = 6	1
Total: 1 x (8+12+6+1) = 27				

1. If other’s bits all 0: 000 (1 case)
2. If other has one bit set: 010, 100, 001 (3 cases)
3. If other has two bits set: 110, 011, 101 (3 cases)
4. If other has three bits set: 111 (1 cases)

1. 27
2. 27
3. 9
4. 1

Total: 27 + 27 + 9 + 1 = 64

3. Practice - Sample Midterm - 2b

```
2b (6 minutes). How many distinct sensible permissions are there?
Explain.
```

- Understand the definition of “sensible” permissions first!
  - Permission: Owner >= Groups >= Others

2. Make use of some mathematical tricks

owner	group	others
_ _ _	_ _ _	_ _ _

- All possibilities for the first bit: (owner bit >= group bit >= others bit)
  - 1-1-1, 1-1-0, 1-0-0, 0-0-0
- Same for the second/third bit
- Total: 4 x 4 x 4 = 64

3. Practice - Sample Midterm - 3

```
3 (4 minutes). Give two good reasons why backups do not suffice for
version control.
```

- No Git/Version control for midterm

3. Practice - Sample Midterm - 4

```
4 (4 minutes).  Give an example of how renaming a dangling symbolic link
can transform it into a non-dangling symbolic link.
```

- What is symbolic link?
  - Symbolic link points to a file name
- What makes the symbolic dangling?
  - If the file name no longer exists
- How to turn it back to non-dangling?
  - Make the file exists!

3. Practice - Sample Midterm - 4

```
4 (4 minutes).  Give an example of how renaming a dangling symbolic link
can transform it into a non-dangling symbolic link.
```

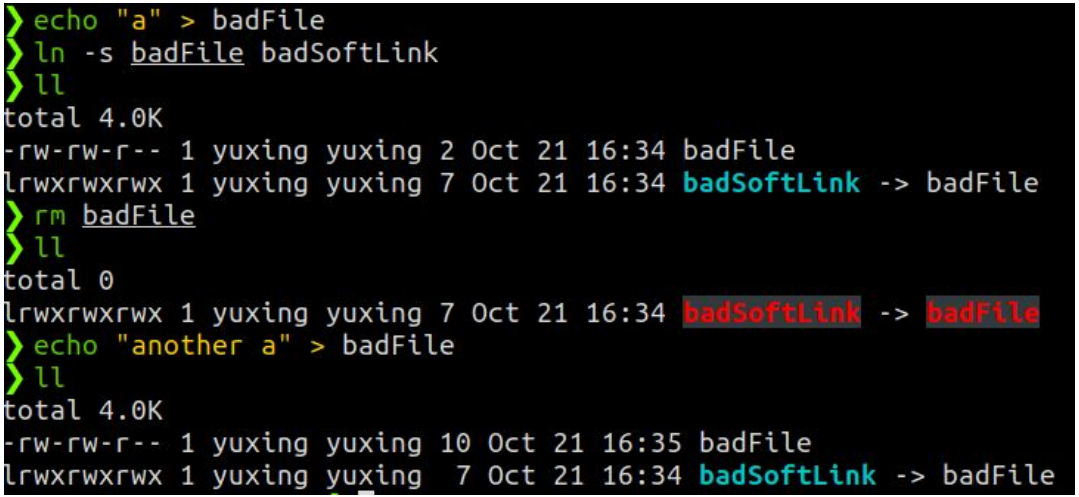
- What is symbolic link?
  - Symbolic link points to a file name
- What makes the symbolic dangling?
  - If the file name no longer exists
- How to turn it back to non-dangling?
  - Make the file exists!

Instruction says:

- **Give an Example**
- **So example!!!**

Example 1

Create the file with the same name



3. Practice - Sample Midterm - 4

```
4 (4 minutes).  Give an example of how renaming a dangling symbolic link
can transform it into a non-dangling symbolic link.
```

- What is symbolic link?
  - Symbolic link points to a file name
- What makes the symbolic dangling?
  - If the file name no longer exists
- How to turn it back to non-dangling?
  - Make the file exists!

Instruction says:

- **Give an Example**
- **So example!!!**

```
❯ mkdir tmp
❯ echo "test" > tmp/badFile
❯ ln -s badFile badSoftLink
❯ ll
total 4.0K
lrwxrwxrwx 1 yuxing yuxing    7 Oct 21 16:39 badSoftLink -> badFile
drwxrwxr-x 2 yuxing yuxing 4.0K Oct 21 16:39 tmp
❯ mv badSoftLink tmp
❯ ll tmp
total 4.0K
-rw-rw-r-- 1 yuxing yuxing 5 Oct 21 16:39 badFile
lrwxrwxrwx 1 yuxing yuxing 7 Oct 21 16:39 badSoftLink -> badFile
```

Example 2

If the soft link is created in the relative path way, we can move the symlink to somewhere target file exists

3. Practice - Sample Midterm - 4

```
4 (4 minutes).  Give an example of how renaming a dangling symbolic link
can transform it into a non-dangling symbolic link.
```

- What is symbolic link?
  - Symbolic link points to a file name
- What makes the symbolic dangling?
  - If the file name no longer exists
- How to turn it back to non-dangling?
  - Make the file exists!

Instruction says:

- **Give an Example**
- **So example!!!**

```
❯ ll
total 4.0K
-rw-rw-r-- 1 yuxing yuxing 2 Oct 21 16:31 badFile
lrwxrwxrwx 1 yuxing yuxing 7 Oct 21 16:31 badSoftLink -> badFile
❯ rm badFile
❯ ll
total 0
lrwxrwxrwx 1 yuxing yuxing 7 Oct 21 16:31 badSoftLink -> badFile
❯ touch anotherFile
❯ ln -sf anotherFile badSoftLink
❯ ll
total 0
-rw-rw-r-- 1 yuxing yuxing  0 Oct 21 16:32 anotherFile
lrwxrwxrwx 1 yuxing yuxing 11 Oct 21 16:32 badSoftLink -> anotherFile
```

Example 3

Make the soft link point to another valid file

3. Practice - Sample Midterm - 5

```
5 (8 minutes). Explain how to arrange for Emacs to treat C-t (i.e.,
control T) as a command that causes Emacs to issue a message like
this:

It is now Tue Apr 27 10:30:34 2020.

in the echo area.  The message should contain the current date and
time.
```

- What we know?
  - How did you add commands in Emacs in you HW?
    - Write a lisp function
    - Load that file to Emacs (`M-x load-file <RET> /path/to/your/file`)
  - What if I don't remember?
    - Look up your printed resources: Emacs Reference Card!
    - Please review your HWs before midterm. (print and bring all HWs with you)

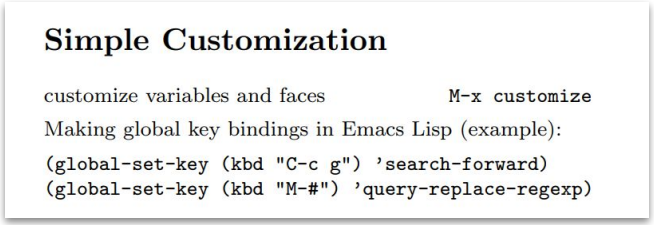
- What we don't know?
  - How to bind the function to a key?
  - How to print the time?

3. Practice - Sample Midterm - 5

5 (8 minutes). Explain how to arrange for Emacs to treat C-t (i.e., control T) as a command that causes Emacs to issue a message like this:

It is now Tue Apr 27 10:30:34 2020.

in the echo area. The message should contain the current date and time.



- What we don't know? (For online exams: Google; For us: the resources you bring to the exam)
  - How to bind the function to a key? (Go to your **Emacs Reference Card**)
    - `M-x global-set-key <RET> C-t print-time <RET>`
  - How to print the time?
    - Google for this online exam
      - `current-time-string`
      - `format-time-string`
    - But for us, we should at least know
      - We will use "`message`" function to print the time string
        - `(message "It is now %s." time-string)`
      - We will call some function to get the time-string

```
(defun print-time ()
  (interactive)
  (message (concat "It is now " (current-time-string) ".")))
(defun print-time ()
  (interactive)
  (message (format-time-string "It is now %a %b %e %T %Y.")))
```

3. Practice - Sample Midterm - 5

```
5 (8 minutes). Explain how to arrange for Emacs to treat C-t (i.e.,
control T) as a command that causes Emacs to issue a message like
this:

It is now Tue Apr 27 10:30:34 2020.

in the echo area.  The message should contain the current date and
time.
```

- Write a lisp function to print time message
- Load that file to Emacs
  - `M-x load-file <RET> /path/to/your/file`
- Bind the function to key "`C-t`"
  - `M-x global-set-key <RET> C-t print-time <RET>`
- Inside the function
  - Print the time string with "`message`"
    - `(message "It is now %s." time-string)`
  - Call some function to get the time-string

3. Practice - Sample Midterm - 6

6 (4 minutes). If a Python class C has a method M and a class variable V, what sort of thing does C.__dict__.items() return and why?

- This is more for exams with computer and Internet
- `C.__dict__.items()` What's that?
  - Write a program for this

```
class C:
    V = 'someVar'

    def M():
        return 1

print(C.__dict__.items())
```

  - Google it

  - We see that `C.__dict__.items()` returns a dict_items object containing (among others) the entries
    - '`V`' → the value we set the class variable `V` to
    - '`M`' → <the function `C.M`>
  - This is because the `__dict__` property on objects contains all of an object's writable attributes, which includes both its data members and methods.

3. Practice - Sample Midterm - 7a

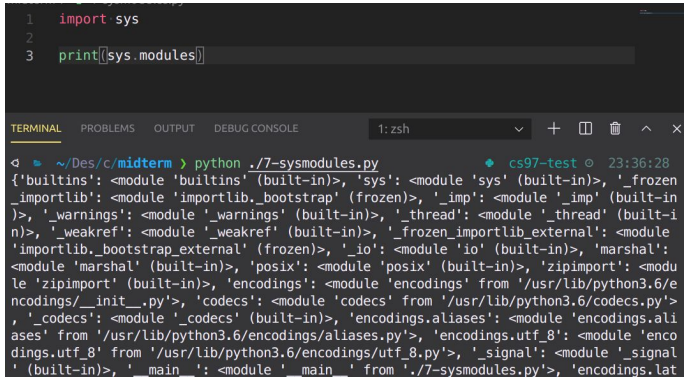
```
7. By default, the Python expression ‘sys.modules’ signals a
NameError, but if you execute ‘import sys’ first the expression does
not signal that error.

7a (2 minutes).  Briefly explain why not.
```

- Modules in Python: That does `import xxx` mean?
- There is no variable named ‘`sys`’ that exists by default.
- Once the `import` statement is executed, the ‘`sys`’ module is imported and bound to the identifier ‘`sys`’.
- Then we can use anything defined in `sys`

3. Practice - Sample Midterm - 7b

```
7b (4 minutes).  Give an example of using the value of the expression
‘sys.modules’ to find out something about your Python process.  Your
example should explore at least one level past the value of
sys.modules itself.
```



- This is a question for online exams
  - Test on your computer or Google it.
- A dict of all imported modules in the current Python process
- This kind of questions shouldn’t appear in paper-based exam
- **Unless Prof. explicitly talked about this on lectures**
- What if it just appears? Analyze through the names:
  - sys => system, some system related information
  - modules => module related information
  - So we can say: sys.modules should give some information related to modules we import to our program; For concrete example, ..😢. (well skip this one if prof never mention any concrete egs)

3. Practice - Sample Midterm - 8

8. Consider the following shell script.

```
#!/bin/sh
atom='[a-zA-Z0-9]+'
string='\\"([^"\]|\\.)*\\"'
word="($atom|$string)"
words="$word(\\.$word)*"
grep -E "$words" | grep ' '
```

8a (4 minutes). Explain briefly what this shell script does, from its user's viewpoint.

Contents

1. Logistic
2. Strategy
3. Overview with Practice
4. **RegEx**

4. RegEx

- Sequence of characters, that specifies a search pattern
- Basic Regular Syntax (BRE) v.s. Extended Regular Syntax (ERE)
  - Different in the behavior of a few special characters like “+”. “?”, parentheses”(“ “)”, braces and “|”
  - BRE: these characters do not have special meaning unless prefixed with backslash \
  - ERE: these characters have special meaning unless prefixed with backslash
  - Resources:
    - https://learnbyexample.github.io/gnu-bre-ere-cheatsheet/
    - https://www.gnu.org/software/sed/manual/html_node/BRE-vs-ERE.html
    - https://en.wikipedia.org/wiki/Regular_expression#:~:text=In%20the%20POSIX%20standard%2C%20Basic,Syntax%20(ERE)%20does%20not.&text=Matches%20the%20starting%20position%20within,starting%20position%20of%20any%20line.

4. RegEx

https://regex101.com/

- Basic match: characters match characters; e.g.: `abc` matches `abc`
- Anchors
  - `^`: matches the start of the string; e.g.: `^the` matches `the line` but not `here's the line`
  - `$`: matches the end of the string; e.g.: `end$` matches `the line end` but not `end the line`
- Quantifiers
  - `*`: matches zero or more; e.g.: `abc*` matches `ab`, `abc`, `abccc`, ...
  - `+`: matches one or more; e.g.: `abc+` matches `abc`, `abccc`, `abccccc`, ...
  - `.`: matches any single character; e.g.: `a.c` matches `abc`, `adc`, `aec`, ...,
    - but `[a.c]` matches only `a, ., or c`
  - `?`: matches zero or one; e.g.: `abc?` matches `ab`, `abc`
  - `{n}`: matches `n` times; e.g.: `abc{2}` matches `abcc`
  - `{n,}`: matches `n` or more times; e.g.: `abc{2,}` matches `abcc, abccc`, ...
  - `{n,m}`: matches `n` to `m` times; e.g.: `abc{2,5}` matches `abcc`, `abccc`, `abcccc`, `abccccc`
  - `()`: specify a sub expression; e.g.: `a(bc)*` matches `a`, `abc`, `abcbcbc`

4. RegEx

- OR operator
  - `|`: means or; e.g.: `a(b|c)` matches `ab`, `ac`
- Grouping
  - `[]`: matches a single character that is contained within the brackets; e.g: `[abc]` matches `a, b, c`; `[a-c]` matches `a, b, c`
  - `[^ ]`: matches a single character that is **NOT** contained within the brackets; e.g: `[^a-z]` matches any single character that is not a lowercase letter from `a` to `z`

3. Practice - Sample Midterm - 8

8. Consider the following shell script.

```
#!/bin/sh
atom='[a-zA-Z0-9]+'
string='\\"([^"\]|\\.)*\\"'
word="($atom|$string)"
words="$word(\\.$word)*"
grep -E "$words" | grep ' '
```

8a (4 minutes). Explain briefly what this shell script does, from its user's viewpoint.

- Figure out what we know
  - atom = regex for alphanumeric character and 1 or more
  - string = ? looks so complicated
  - word = regex: atom or string
  - words = word + optional extra stuff
  - grep = works like a filter, ensure the output fits words with a space

- It prints any line that satisfy both of the following:
  - Contains a space
  - Contains at least one of:
    - An alphanumeric character ($atom) or
    - A sequence of ($string)
- If you have more time, figure out what string is:
  - A (possibly empty) sequence of non-‘"’ (literal double-quote) characters that is surrounded by ‘\"’ (literal backslash followed by literal double-quote). If ‘\’ (literal backslash) appears within the string, then it must start a well-formed escape sequence (it has another character that follows it) before the closing ‘\"’ ($string).

3. Practice - Sample Midterm - 8b

```
8b (3 minutes).  How would the script's behavior change if you removed
the '-E' from this script?
```

- If we were to remove the '-E' flag, then POSIX Basic Regular Expression (BRE) is used instead of the Extended Regular Expression (ERE).
- This means that many meta-characters used, including the '(', '|', and ')' in $word, are treated as normal characters to match rather than the start of group, disjunction, and end of a group, respectively.

3. Practice - Sample Midterm - 8c

8c (5 minutes). Modify the original script so that calls grep just once instead of twice, without changing the script's I/O behavior.

8. Consider the following shell script.

```
#!/bin/sh
atom='[a-zA-Z0-9]+'
string='\\"([^"\]|\\.)*\\"'
word="($atom|$string)"
words="$word(\\.$word)*"
grep -E "$words" | grep ' '
```

8a (4 minutes). Explain briefly what this shell script does, from its user's viewpoint.

- Many ways to achieve that!

  E.g.:

```
space=" "
grep -E "$words$space|$space$words"
```

What if spaces appear inside "words" as there are many " 's in the definition.
Maybe more:

```
string_with_space='\\"([^"\]|\\[ ])*\\"' #substitute "." in string def to "[ ]"
# then define another word, substitute string by string_with_space
word_with_space="($atom|$string_with_space)"
words_with_space=
"$word(\\[ ]$word)*|$word_with_space(\\[ ]$word)*"
grep -E "$words$space|$space$words|$words_with_space"
```

Or use another command that has similar functionally as "grep": maybe "sed"

```
grep -E "$words" | sed -n '/ /p'
```