Week 2

CS 97 Disc 1A

Today’s Plan

- Poll
- Emacs Lisp 101
- Assignment 2 Lab hints
- Assignment 2 HW hints
- Poll
- LA Activities

Emacs Lisp 101

Basics

- Lisp is a strange programming language
- Lisp stands for LISt Processing
  - handles lists (and lists of lists) by putting them between parentheses
  - the parentheses mark the boundaries of the list

Lists

- '(vanilla chocolate matcha caramel)
- '(this list has (a list inside of it))
- (+ 2 2)
- **Atoms** are the numbers, words, and symbols that are used to build the lists
  - It cannot be further divided into smaller parts and still mean the same thing as part of a program
- Different kinds of atoms
  - Numbers- 23, 509, 12301
  - Symbols- +, foo, forward-line, vanilla
  - Strings- “hello”, “bye”

Evaluating

- Any list is a program ready to run (or evaluate)
- The single apostrophe ', known as quote, tells Lisp to do nothing to the list other than take it as it is written
- Otherwise, the first item of the list is special- a command for the computer to obey
  - (+ 2 2)
- (+ 2 (+ 3 3))?

How to run?

- *scratch* buffer is provided in Emacs for evaluating Emacs Lisp expressions interactively
- Use M-: (or M-x eval-expression) to read a single Emacs Lisp expression in the minibuffer, evaluate it, and print the value in the echo area
- Place your cursor immediately after the right hand parenthesis of the following list and then type C-x C-e (eval-last-sexp)

Arguments

- (+ 2 2)
  - The numbers added by + are arguments
- The arguments’ data type is dependent on the function
  - (+ 2 2)
  - (concat "water" "melon")
  - (substring "This class is great" 5 10)

message

- Sends messages to the user by printing them in the echo area
  - (message "How you doin’?")
- (message "The name of this buffer is: %s." (buffer-name))
  - %d looks for a number
  - %s looks for a sting

Setting value of a variable

- (set 'flowers '(rose violet daisy buttercup))
- (setq flowers '(rose violet daisy buttercup))
- Counting
  - (setq counter 0)
  - (setq counter (+ counter 1))

Buffer

- Buffer is an object that hold the text you are editing in Emacs
  - Each time you visit a file, a buffer is used to hold the file's text.
- (buffer-size)
  - tells you the size of the current buffer
- (point)
  - the current position of the cursor
- point-min and point-max
  - the value of the minimum (and maximum) permissible value of point in the current buffer
  - point-min should be 1 with no narrowing

Optional What is narrowing?

- Narrowing is a mechanism whereby you can restrict yourself, or a program, to operations on just a part of a buffer.
- Narrowing can make it easier to concentrate on a single subroutine or paragraph by eliminating clutter.

defun

- Function definition
- A function definition has up to five parts following the word defun:
  - The name of the symbol to which the function definition should be attached.
  - A list of the arguments that will be passed to the function. If no arguments will be passed to the function, this is an empty list, ().
  - Documentation describing the function. (Technically optional, but strongly recommended.)
  - Optionally, an expression to make the function interactive so you can use it by typing M-x and then the name of the function; or by typing an appropriate key or keychord.
  - The code that instructs the computer what to do: the body of the function definition.

```
(defun multiply-by-seven (number)
  "Multiply NUMBER by seven."
  (* 7 number))
```

let

- Attach or bind a symbol to a value in such a way that the Lisp interpreter will not confuse the variable with a variable of the same name that is not part of the function
- Prevents confusion
  - Creates a name for a local variable that overshadows any use of the same name outside the let expression
  - Local variables created by a let expression retain their value only within the let expression itself
- Uninitialized variables are set to nil
  - Nil means empty list or false

let

```
(let ((variable value)
      (variable value)
       …)
  body…)
```

```
(let ((class "cs97")
       (mood "happy"))
  (message "%s makes me very %s." class mood))
```

if

```
(if (= 3 (- (+ 2 2) 1))                                          ; if-part
    (message "2 plus 2 is 4, minus 1 that’s 3 quick mafs"))   ; then-part
```

```
(if (> 4 5)                                          ; if-part
  (message "4 falsely greater than 5!")   ; then-part
  (message "4 is not greater than 5!"))   ; else-part
```

save-excursion

- It saves the location of point, executes the body of the function, and then restores point to its previous position if its location was changed.
- Its primary purpose is to keep the user from being surprised and disturbed by unexpected movement of point.

save-restriction

- Executes the code in the body of the save-restriction expression, and then undoes any changes to narrowing that the code caused
- If the buffer is narrowed and the code that follows save-restriction gets rid of the narrowing, save-restriction returns the buffer to its narrowed region afterwards
- Any narrowing the buffer may have is undone by the widen command that immediately follows the save-restriction command

Lab 2.1

- Using *scratch*
  - C-x b
  - Evaluate- LFD (or C-j): linefeed, a key similar to return
- Example: 5**(2**3) → (expt 5 (expt 2 3))
- Maybe use < or > to check if the number is less than the max of 64-bit signed integer?
- cycle-spacing

Lab 2.2- Scripting Emacs

- Relevant expressions
  - (what-line) - gives the current line number
  - (point-min) - gives the index for the starting point/cursor
  - (point-max) - gives the index for the last point/cursor
  - **(count-lines (point-min) (point-max)) - gives the number of lines between 1st and 2nd point indices**
  - (char-before (point-max)) - gives the char code just before the arg point index
  - char code for newline char (or line feed) is 10

```
(defun count-lines (start end)
  "Return number of lines between START and END.
This is usually the number of newlines between them,
but can be one more if START is not equal to END
and the greater of them is not at the start of a line."
```

```
(defun line-number-at-pos (&optional pos absolute)
  "Return buffer line number at position POS.
If POS is nil, use current buffer location.

If ABSOLUTE is nil, the default, counting starts
at (point-min), so the value refers to the contents of the
accessible portion of the (potentially narrowed) buffer.  If
ABSOLUTE is non-nil, ignore any narrowing and return the
absolute line number."
```

Lab 2.2- Scripting Emacs

```
(defun what-line ()
  "Print the current buffer line number and narrowed line number of point."
  (interactive)
  (let ((start (point-min))
        (n (line-number-at-pos)))
    (if (= start 1)
        (message "Line %d" n)
      (save-excursion
        (save-restriction
          (widen)
          (message "line %d (narrowed line %d)"
                   (+ n (line-number-at-pos start) -1) n))))))
```

Homework- Python scripting

- wget https://web.cs.ucla.edu/classes/fall20/cs97-1/assign/randline.py
- python randline.py /dev/null
  - What happens?
- python randline.py [INSERT FILE HERE]
  - What happens?
- python randline.py -n 30 [INSERT FILE HERE]
  - What happens?
- python3 randline.py [INSERT FILE HERE]
  - What happens?

randline.py

```
#!/usr/bin/python

import random, sys
from optparse import OptionParser


class randline:
    def __init__(self, filename):
        f = open(filename, 'r')
        self.lines = f.readlines()
        f.close()

    def chooseline(self):
        return random.choice(self.lines)

def main():
    version_msg = "%prog 2.0"
    usage_msg = """%prog [OPTION]... FILE

Output randomly selected lines from FILE."""

    parser = OptionParser(version=version_msg,
                          usage=usage_msg)
    parser.add_option("-n", "--numlines",
                      action="store", dest="numlines", default=1,
                      help="output NUMLINES lines (default 1)")
    options, args = parser.parse_args(sys.argv[1:])
```

Tells the shell which interpreter to use

Import statements, similar to include statements

The beginning of the class statement: randline

- The constructor
  - Creates a file handle
  - Reads the file into array of strings called lines
  - Close the file

- The beginning of a function belonging to randline: chooseline
  - Randomly select a number between 0 and the size of lines
  - Returns the line corresponding the randomly selected number

Main function

- Some basic information about the command

- Create an instance of OptionParser

- Add option -n or --numlines, store the value of the option in numlines
  If this option is not in the command line, then the default value is 1

- Parse the command line starting from after sys.argv[0], which is the script name python, and store the options and arguments

randline.py

```
....
    try:
        numlines = int(options.numlines)
    except:
        parser.error("invalid NUMLINES: {0}".
                     format(options.numlines))
    if numlines < 0:
        parser.error("negative count: {0}".
                     format(numlines))
    if len(args) != 1:
        parser.error("wrong number of operands")
    input_file = args[0]

    try:
        generator = randline(input_file)
        for index in range(numlines):
            sys.stdout.write(generator.chooseline())
    except IOError as (errno, strerror):
        parser.error("I/O error({0}): {1}".
                     format(errno, strerror))

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

Convert the value from the -n option (default 1) to integer and store in numlines

If the value is invalid, ERROR

If the numlines is less than 0, ERROR

If the number of arguments is not 1, ERROR

Set input_file as the first argument (the only argument)

Create object generator, which is an instance of randline class, and pass in input_file

Loop for the numlines times (if numlines = 4, loop 4 times)
Write the randomly chosen line to standard output

Raise error related to input/output operation
Errno is the error code and strerror is the error message to the error code

Run main()

*Every module in Python has a special attribute called __name__. The value of __name__ attribute is set to “__main__” when module is run as main program. Otherwise, the value of __name__ is set to contain the name of the module.

shuf.py

- Probably need a class like randline?
  - Maybe store a list of lines and the number of lines?
- Add all these option: --echo (-e), --input-range (-i), --head-count (-n), --repeat (-r), and --help
  - import argparse
  - parser = argparse.ArgumentParser()
  - parser.add_argument("-r","--repeat",action="store_true",dest="Replace", default=False, help="Repeat output values, that is, select with replacement\n")
- Use if-else and try-except statements to cover different cases?

Example: -i/--input-range option

```
parser.add_argument("-i", "--input-range",
                        action="store", dest="input_range", default=None,
                        type="string",
                        help="Act as if input came from a file containing the
range of unsigned decimal integers lo…hi, one per line\n")
…

option, argument = parser.parse_args(sys.argv[1:])
```

Stored in option.input_range

Example: -i/--input-range option

```
If what's stored is not None
     If the number of argument is not 0
          Then there's an ERROR!

     Split what's stored in option_ir (should have two things,
     one on the left and one on the right o '-'

     If there's only one thing, then ERROR!

     Let's try convert lo to an integer, and if that fails, ERROR!

     Let's try convert hi to an integer, and if that fails, ERROR!

     Check if lo is actually smaller than hi, if lo and hi
     don't make sense, ERROR!

     Okay, seems like it's fine so we proceed

        if LO > HI+1:

Else blah blah blah
```