Patching

and

More Python

CS 35L

Spring 2018 - Lab 3

Assignment 7 - Reminder

For assignment 7, you will need a Seeed Studio BeagleBone Green Wireless Development Board

Get it sooner rather than later!

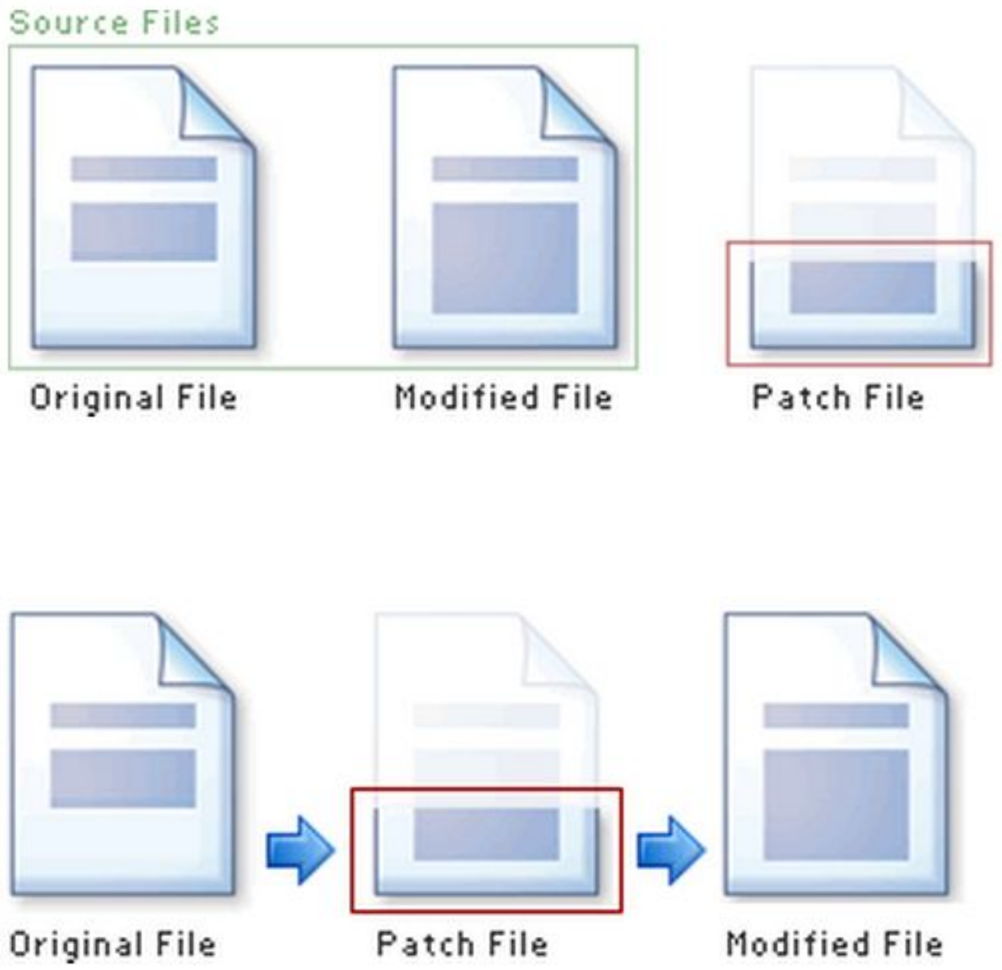
See the specs for assignment 7 for details: https://web.cs.ucla.edu/classes/winter18/cs35L/assign/assign7.html

Patching

Patching

- A patch is a piece of software designed to fix problems with or update a computer program.
- It’s a `diff` file that includes the changes made to a file
- A person who has the original (buggy) file can use the `patch` command with the diff file to add the changes to their original file

Applying a Patch



diff Unified Format

- --- path/to/original_file
- +++ path/to/modified_file

- @@ -l,s +l,s @@
  - @@: beginning and end of a hunk
  - l: beginning line number
  - s: number of lines the change hunk applies to for each file
  - A line with a:
    - - sign was deleted from the original
    - + sign was added in the new file
    - ‘ ‘ stayed the same

Patching

- `cd` into directory patch considers pwd
- vim or emacs patch_file: copy and paste the patch content
- `patch [options] [originalfile [patchfile]]`
- `patch -p`**`num`**` <patch_file`
- *`man patch`* to find out about p**num**
- BE AWARE: p**num** defaults to p1 if omitted
- `cd` into the coreutils-7.6 directory and type `make` to rebuild patched `ls.c`
- More patch command examples - link

More Python

What is Python?

- Not just a scripting language
- Object-Oriented language
  - Classes
  - Member functions
- Compiled and interpreted
  - Python code is compiled to bytecode
  - Bytecode interpreted by Python interpreter
- Not as fast as C but easy to learn, read and use

Indentation

- Python has **no braces** or keywords for code blocks
  - C delimiter: {}
  - bash delimiter:
    - then…else…fi (if statements)
    - do…done (while, for loops)
- **Indentation** makes all the difference
  - Tabs change code's meaning!!

Python List

- Common data structure in Python
- A python list is like a C array but much more:
  - **Dynamic**: expands as new items are added
  - **Heterogeneous:** can hold objects of different types
- How to access elements?
  - List_name[index]

Example

- `>>> t = [123, 3.0, ‘hello!’]`
- `>>> print t[0]`
  - 123
- `>>> print t[1]`
  - 3.0
- `>>> print t[2]`
  - hello!

List Operations

- >>> list1 = [1, 2, 3, 4]
- >>> list2 = [5, 6, 7, 8]
- Adding an item to a list:
  - list1.append(5)
  - **Output: [1, 2, 3, 4, 5]**
- Merging lists:
- >>> merged_list = list1 + list2
- >>> print merged_list
  - **Output: [1, 2, 3, 4, 5, 5, 6, 7, 8]**

for loops

```
list = [‘Mary’, ‘had’, ‘a’, ‘little’, ‘lamb’]
```

```
for item in list:
   print item
```

Result:

```
Mary
had
a
little
lamb
```

```
for i in range(len(list)):
   print i
```

Result:

```
0
1
2
3
4
```

Functions

```
def hello(strange, world="interesting"):
    print("hello " + strange)
    print(world)

hello("class")
hello(world="python",strange="everybody")
```

Argparse Library

- Powerful library for parsing command-line options (update of older optparse library)
  - **Argument:**
    - String entered on command line and passed to script
    - Elements of sys.argv[1:] (sys.argv[0] is program name)
  - **Option:**
    - An argument that supplies extra information to customize the execution of a program
  - **Option Argument:**
    - An argument that follows an option and is closely associated with it. It is consumed from the argument list when the option is

Python Walk-Through

```
#!/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."""
```

Tells the shell which interpreter to use

Import statements, similar to include statements
Import OptionParser class from optparse module

The beginning of the class statement: randline
    The constructor
        Creates a file handle
        Reads the file into a list called lines
        Close the file

The beginning of a function belonging to randline
Randomly select an element from self.lines and return it

The beginning of main function
version message
usage message

Python Walk-Through

```
parser = OptionParser(version=version_msg,
      usage=usage_msg)
```

Creates OptionParser instance

```
parser.add_option("-n", "--numlines",
      action="store", dest="numlines",
      default=1, help="output NUMLINES
      lines (default 1)")
```

Start defining options, action “store” tells optparse to take next argument and store to the right destination which is “numlines”. Set the default value of “numlines” to 1 and help message.

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

options: an object containing all option args
args: list of positional args leftover after parsing options

```
try:
    numlines = int(options.numlines)
except:
    parser.error("invalid NUMLINES: {0}".
      format(options.numlines))
```

Try block
  get numline from options and convert to integer
Exception handling
  error message if numlines is not integer type, replace {0 } w/ input

```
if numlines < 0:
    parser.error("negative count: {0}".
format(numlines))
```

If numlines is negative
  error message

```
if len(args) != 1:
    parser.error("wrong number of operands")
input_file = args[0]
```

If length of args is not 1 (no file name or more than one file name)
  error message
Assign the first and only argument to variable input_file

```
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))
```

Try block
  instantiate randline object with parameter input_file
  for loop, iterate from 0 to numlines – 1
    print the randomly chosen line
Exception handling
  error message in the format of “I/O error (errno):strerror

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

In order to make the Python file a standalone program

Running Python scripts

- Download `randline.py` from assignment website
- Make sure it has executable permission:

  chmod +x randline.py
- Run it, for example

  ./randline.py –n 4 filename

  n: is an option indicating the number of lines to write

  4: is an argument to n (you can use any integer number)

  Filename: is a program argument

Python2 vs Python3

- Python2
  - First released in 2000. Final major release in 2010.
  - Considered a legacy language by many.
  - Slightly better library support (as it’s older).
- Python3
  - First released in 2008. Major releases are ongoing.
  - Considered the present and future of python.
  - More limited library support, as it’s newer.

For a reasonably readable rundown of the language differences, see this blog post.

Homework 3 - Overview

- randline.py script
  - Get some familiarity with python by reading the script.
  - Answer a few questions about it.
- Implement the **`shuf`** command in python
- Port your `shuf` script from python2 to 3

shuf.py

- Support the following options
  - --echo (-e), --head-count (-n), --repeat (-r), and --help
- Support variable number and types of arguments
  - File names and – for stdin
- Change usage message to describe behavior
- Port `shuf.py` to Python 3
- `man shuf` or online docs for more details
- Read coreutils shuf source if you’re confused

Homework 3 Hints

- Read first 9 chapters here: docs.python.org/3.6/tutorial/
- Q4: Python 3 vs. Python 2
  – Look up “automatic tuple unpacking”
- Use `python` in shell for Python 2
- Use `python3` in shell for Python 3

```
which python3  →  /usr/local/cs/bin/python3
```