CS35L – Spring 2019

Slide set:	3.2
Slide topics:	Modifying and Rewriting Software, Python
Assignment:	3

Functions

A function is a block of organized, reusable code that is used to perform a single, related action. They provide better modularity for your application and a high degree of code reusing.

Syntax:

```
def function_name( parameters ):

        #code inside the function
```

Functions examples

Example 1:

```
def printme(new_string): #string is a parameter
        #This prints a passed string into this function
        print new_string
        return
```

Example 2: To print sum of numbers in a list

```
def find_sum(new_list):
        sum=0  #initialize variable*
        for element in new_list:
                sum = sum + element
  return sum #returns the computed sum

answer_variable=find_sum([2,3,4,5]) #function call
print answer_variable
```

Note:

```
#this is a single-line comment

"'this is a
multi-line
Comment"'
```

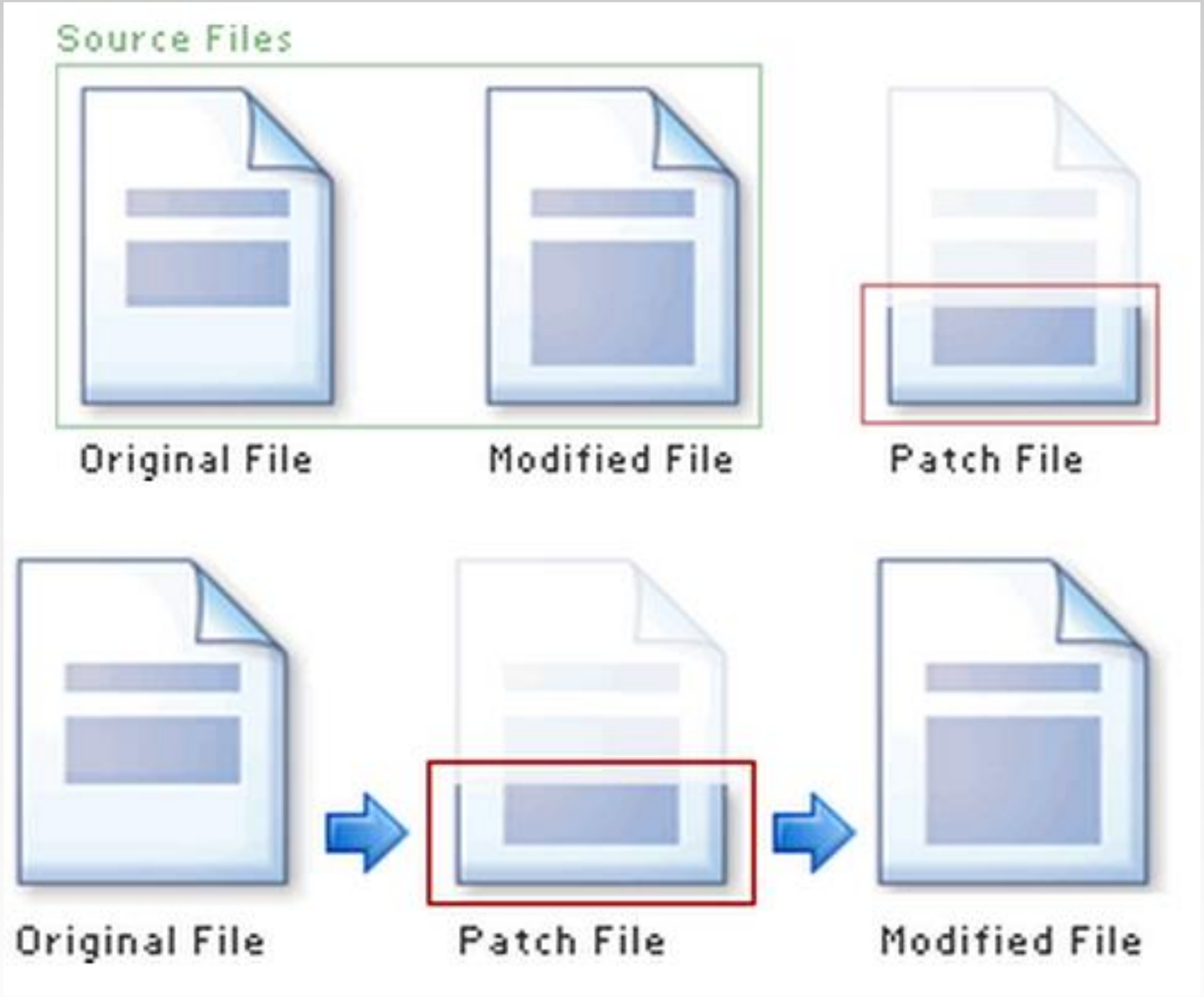
Classes in Python

```
class Rectangle:
    def __init__(self, x, y):
        self.l = x
        self.b = y

    def getArea(self):
        return self.l * self.b

    def getPerimeter(self):
        return 2 * (self.l + self.b)

def main():
    rect = Rectangle(3, 4)
    print("Area of Rectangle:", rect.getArea())
    print("Perimeter of Rectangle:", rect.getPerimeter())

main()
```

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

- diff –u original_file modified_file

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

- @@ -l,s +l,s @@
  - @@: beginning 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 to the original
    - stayed the same

Task 1

Take a list a = [1,1,2,3,5,8,13,21,34,55,89] and write a program that prints out all the elements of the list that are less than 5.

Instead of printing the elements one by one, make a new list that has all the elements less than 5 from this list in it and print out this new list.

Ask the user for a number and return a list that contains only elements from the original list a that are smaller that that number given by the user.

Task 2

Write a program that accepts sequence of newlines from the user as input till the user gets bored. (The user gets bored when he/she presses Enter twice)

Hint: to detect if Enter is pressed twice (in other terms - the user input is void)

Suppose your input string name is 's' just type:

```
if s:
        #write code to capitalize (s.upper()) this input string 's' and append it
        #to a  'new_list'
```

Now, print all the elements of this 'new_list'

Suppose the following input is supplied to the program:

Hello world

Practice makes perfect

Task 3

Write a Python program to return a string containing the first 2 and the last 2 chars from a given a string.

Is it possible to do so with just a single line of code?

Sample String : 'w3resource'

Expected Result : 'w3ce'

Sample String : 'w3'

Expected Result : 'w3w3'

Task 4

Create a python dictionary with the following keys and values:

“Names” : [“Mickey”, “Minnie”]

“Mickey” : [“UCLA”, “Bachelor Degree”]

“Minnie” : [“UCB”, “Bachelor Degree”]

The values in the dictionary are in the form of a list.

- Now traverse the list whose key is ‘Names’ and for every element in this list, find the corresponding key (eg. ‘Mickey’). Append the word “Computer Science” to the value (eg. the list of ‘Mickey’) of that particular key.

- Now create a new key-value pair for “DonaldDuck” - [“Stanford”, “PhD”, “Computer Science”]. Add the name ‘DonaldDuck’ to the ‘Names’ list as well.

Lab 3

- Coreutils 8.29 has a problem
  - $ la –A is equivalent to ls -a –A
  - if the current directory has two files named .foo and bar, the command la -A outputs four lines, one each for ., .., .foo, and bar.
  - These users want la -A to output just two lines instead, one for .foo and one for bar
- Why?
  - the -a option always overrides the -A option regardless of which option is given first
- Want the flag that comes later to take effect
- Fix the ls program

Getting Set Up (Step 1)

- Download coreutils-8.29 to your home directory
  - Use ‘wget’
- Untar and Unzip it
  - tar –xJvf coreutils-8.29.tar.xz
- Make a directory ~/coreutilsInstall in your home directory (this is where you’ll be installing coreutils)
  - mkdir ~/coreutilsInstall

Building coreutils (Step 2)

- Go into coreutils-8.29 directory. This is what you just unzipped.
- Read the INSTALL file on how to configure "make", especially **--prefix** flag
- Run the configure script using the prefix flag so that when everything is done, coreutils will be installed in the directory ~/coreutilsInstall
- Compile it: make
- Install it: make install (won't work on Linux server without proper prefix!)
  - Why?

Reproduce Bug (Step 3)

- Reproduce the bug by running the version of ‘ls -a -A ’ in coreutils 8.29
- If you just type $ ls at CLI it won’t run ‘ls’ in coreutils 8.29
  - Why? Shell looks for /bin/ls
  - To use coreutils 8.29: $ ./ls
    - This manually runs the executable in this directory

Patching and Building (Steps 4 & 5)

- cd coreutils-8.29
- vim or emacs patch_file: copy and paste the patch content
- `patch` –**pnum** < patch_file
  - '`man patch`' to find out what **pnum** does and how to use it
- `cd` into the coreutils-8.29 directory and type make to rebuild patched ls.c.
  - Don't install!!

Testing Fix (Step 6)

- Test the following:
  - Modified ls works
  - Installed unmodified ls does NOT work
- Test on:
- Empty directory
  - Directory containing a hidden file
    - With just –a, with just –A
    - With –aA
    - With –Aa
- Answer Q1 and Q2

Optparse Library

(powerful library for parsing command-line options)

- **Argument:**
  - String entered on the command line and passed in to the script
  - Elements of sys.argv[1:] (sys.argv[0] is the name of the program being executed)
- **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.
- Refer to these links:
  - https://docs.python.org/3/howto/argparse.html
  - https://docs.python.org/3/library/optparse.html

Homework 3

- randline.py script
  - Input: a file and a number *n*
  - Output: *n* random lines from *file*
  - Get familiar with language + understand what code does
  - Answer some questions about script (Q3, Q4)
- Implement shuf utility in python

Running randline.py

- Run it
  - ./randline.py –n 3 filename (need execute permission)
  - python randline.py –n 3 filename (no execute permission)

- randline.py has 3 command-line arguments:
- ◦ filename: file to choose lines from
  - ◦ **argument** to script
  - n: specifies the number of lines to write
    - **option**
  - 3: number of lines
    - **option argument** to n

- Output: 3 random lines from the input file

shuf.py

Support the options for shuf

- --echo (-e)
- --head-count (-n)
- --repeat (-r)
- --help

Support all type of arguments

- File names and – for stdin
- Any number of non-option arguments

Error handling

Homework 3

- shuf.py – this should end up working almost exactly like the utility ‘shuf’
  - Check $ man shuf for extensive documentation
- Use randline.py as a starting point!
  - Modify to accomplish logical task of shuf
- shuf C source code :
  - Present in coreutils
  - This will give you an idea of the logic behind the operation that shuf executes
- Python argparse module instead of optparse:
  - How to add your own options to the parser
  - -e -n --repeat --echo etc

Homework 3 Hints

- If you are unsure of how something should be output, run a test using existing shuf utility!
  - Create your own test inputs
- The shuf option --repeat is Boolean
  - Which action should you use?
- Q4: Python 3 vs. Python 2
  - Look up “automatic tuple unpacking”
- Python 3 is installed in /usr/local/cs/bin
  - export PATH=/usr/local/cs/bin:$PATH

Python Walk-Through

```python
#!/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
Create a file handle
Read the file into a list of strings called lines
Close the file

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

The beginning of main function
version message
usage message

Python Walk-Through

```
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:])

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()
```

Creates OptionParser instance

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: an object containing all option args

args: list of positional args leftover after parsing options

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 is negative

  error message

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 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

In order to make the Python file a standalone program