CS 35L- Software Construction Laboratory

Winter 2019

TA: Guangyu Zhou

Lab 3

Course Information

- Assignment 10 presentation starts next Monday
  - **Submit your slides to CCLE Lab 3 Presentation Slides under assignment folder before your presentation**
  - Grading rules
    - 1st unexcused reschedule: -20% points
    - 2nd time: get 0 for assignment 10
    - **Specs: Organization, Subject Knowledge, Graphics, Interaction, Time management**
  - **Participation:**
    - **Extra credit for asking questions for each presentation:**
    - **+1%, +2% … +5% (max) for assignment 10 grade.**

Review: Build Process & Patching

- **configure**
  - Script that checks details about the machine before installation
    - Dependency between packages
  - configure --prefix="absolute/path/to/your/file/"
  - Creates 'Makefile'
- **make**
  - Requires 'Makefile' to run
  - Compiles all the program code and creates executables in current temporary directory
- **make install**
  - make utility searches for a label named install within the Makefile, and executes only that section of it
  - executables are copied into the final directories (system directories)
- **Patch command**

  Usage: patch pNum -i patchfile.diff

Review: Python basics

- Compiled vs. interpreted language; Python vs. others
- Basic data types
- Python variable & assignment
- Mutability: Tuples vs. Lists
- Python control flows
- Python functions & modules

Introduction to Python 2.x II

- Understanding Reference Semantics
    - Assignment of immutable vs mutable types
- More about Python List
- Classes and Objects
- Misc. File I/O, Strings, Exceptions...
- Example

Understanding Reference Semantics I

- **Assignment manipulates references**

—x = y **does not make a copy** of the object y references

—x = y makes x **reference** the object y references

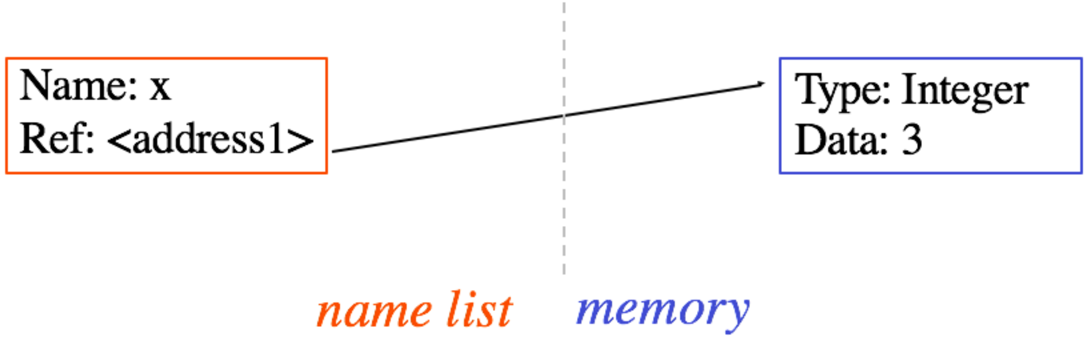
- **Very useful; but beware!**
- **Example:**

```
>>>a=[1,2,3]
>>> b = a
>>> a.append(4)
>>> print b
[1, 2, 3, 4]
```

- **Why??**

Understanding Reference Semantics II

- There is a lot going on when we type: x = 3
- First, an integer *3* is created and stored in memory
- A name *x* is created
- An *reference* to the memory location storing the *3* is then assigned to the name *x*
- So: When we say that the value of *x* is *3*, we mean that *x* now refers to the integer *3*



Understanding Reference Semantics III

- The data 3 we created is of type integer. In Python, the datatypes integer, float, and string (and tuple) are “immutable.”
- This doesn’t mean we can’t change the value of x, i.e. *change what x refers to* …
- For example, we could increment x:

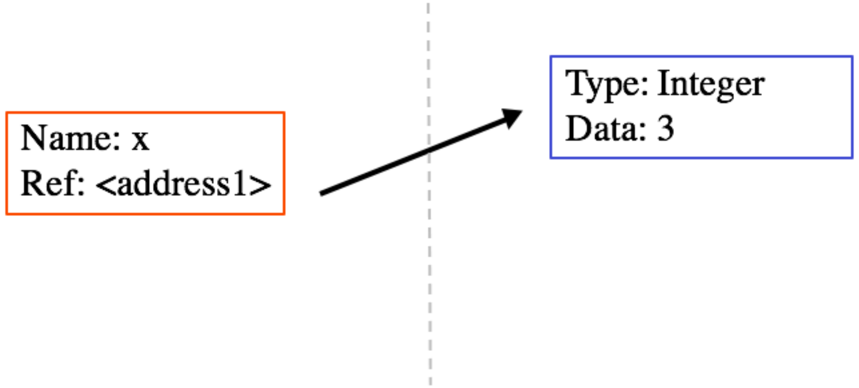
```
>>> x = 3
>>> x = x + 1
>>> print x
4
```

Understanding Reference Semantics IV

- **If we increment x, then what's really happening is:**

1. *The reference of name **x** is looked up.*
2. *The value at that reference is retrieved.*

```
>>>  x = x + 1
```



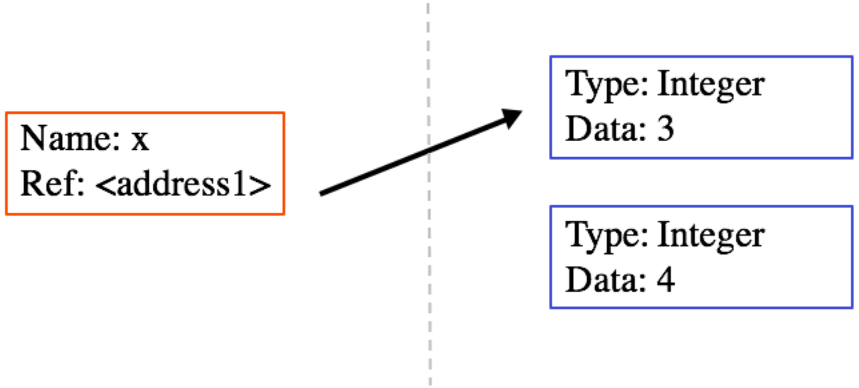
Name: x
Ref: <address1>

Type: Integer
Data: 3

Understanding Reference Semantics IV

- **If we increment x, then what's really happening is:**

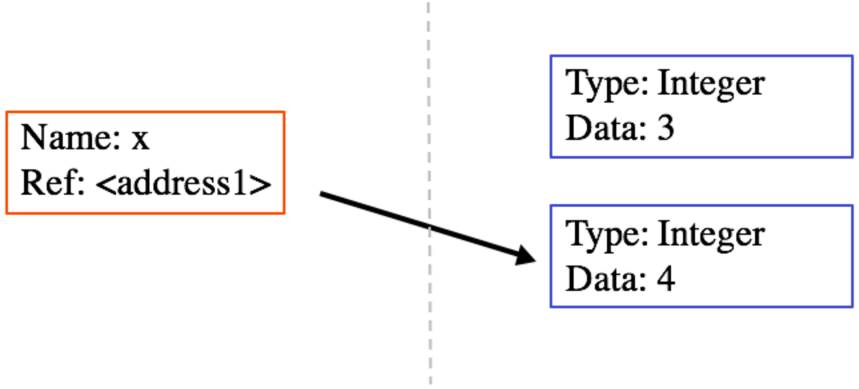
1. *The reference of name* ***x*** *is looked up.*
2. *The value at that reference is retrieved.*
3. *The 3+1 calculation occurs, producing a new data element* ***4*** *which is assigned to a fresh memory location with a new reference.*



Understanding Reference Semantics IV

- **If we increment x, then what's really happening is:**

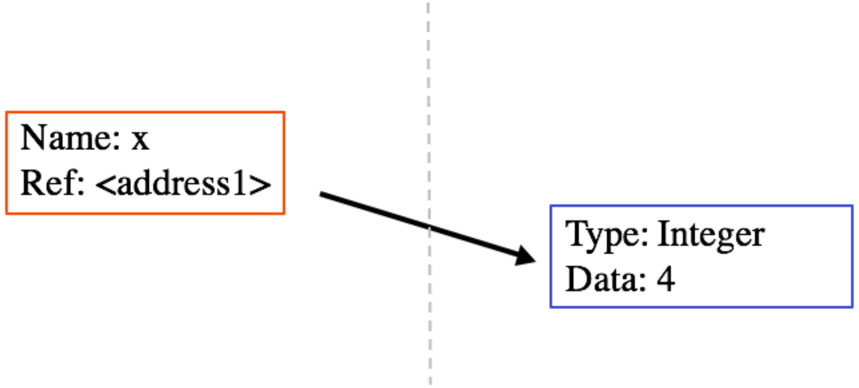
1. *The reference of name **X** is looked up.*
2. *The value at that reference is retrieved.*
3. *The 3+1 calculation occurs, producing a new data element **4** which is assigned to a fresh memory location with a new reference.*
4. *The name **X** is changed to point to this new reference.*



Understanding Reference Semantics IV

- **If we increment x, then what's really happening is:**

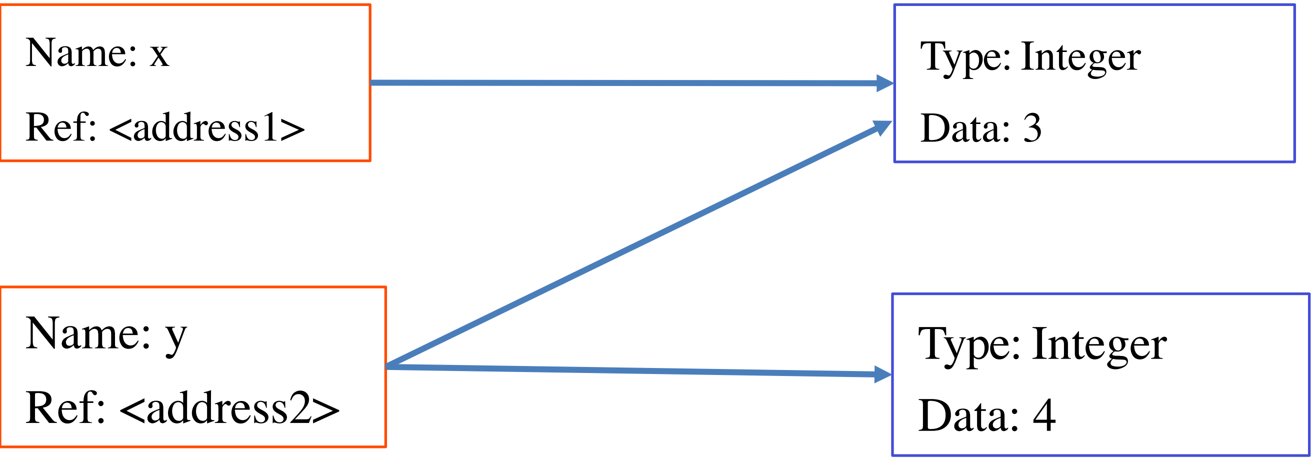
1. *The reference of name* ***x*** *is looked up.*
2. *The value at that reference is retrieved.*
3. *The 3+1 calculation occurs, producing a new data element* ***4*** *which is assigned to a fresh memory location with a new reference.*
4. *The name* ***x*** *is changed to point to this new reference.*
5. *The old data* ***3*** *is garbage collected if no name still refers to it.*



Assignment of immutable vs mutable types

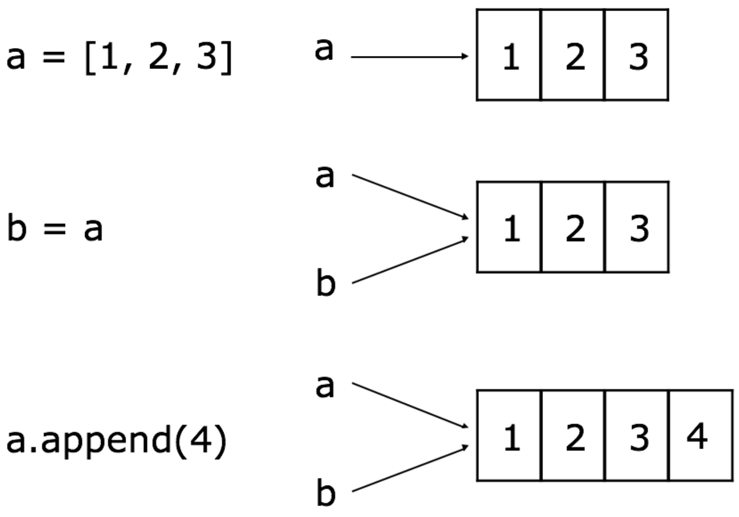
- So, for simple built-in **immutable** datatypes (integers, floats, strings), assignment behaves as you would expect:

```
>>> x = 3          # Creates 3, name x refers to 3
>>> y = x          # Creates name y, refers to 3.
>>> y = 4          # Creates ref for 4. Changes y.
>>> print x        # No effect on x, still ref 3.
3
```



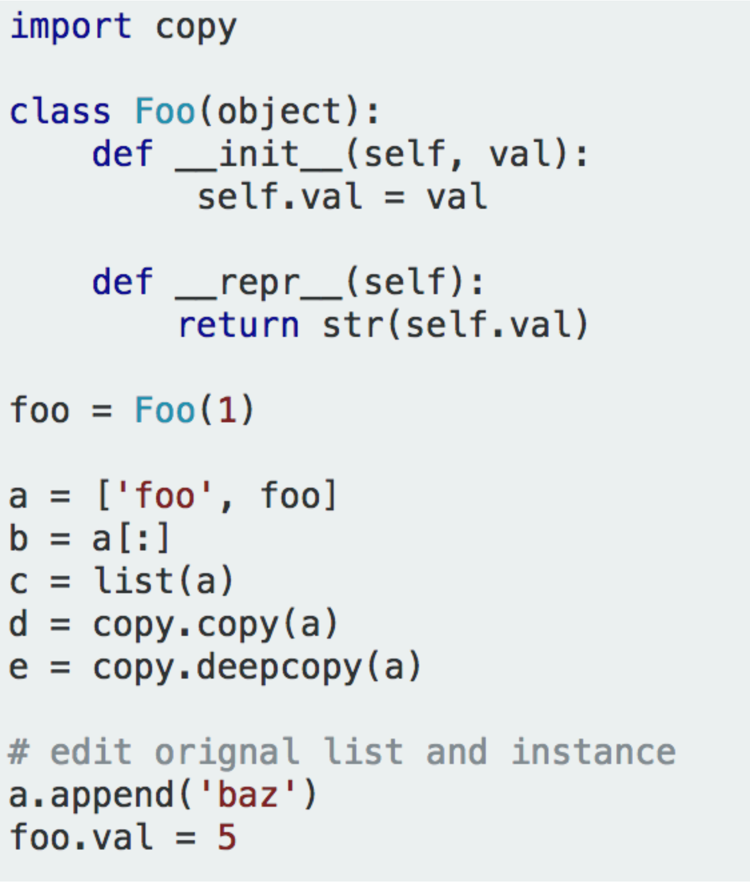
- For other **mutable** data types (lists, dictionaries, user-defined types), assignment works differently.
  - When we change these data, we do it in place. We don't copy them into a new memory address each time.

```
>>> a = [1, 2, 3]
>>> b = a
>>> a.append(4)
>>> print b
[1, 2, 3, 4]
```



More about Python List

- How do we actually copy a list?
  1. Slicing
  2. list()
  3. copy.copy()
  4. copy.deepcopy()

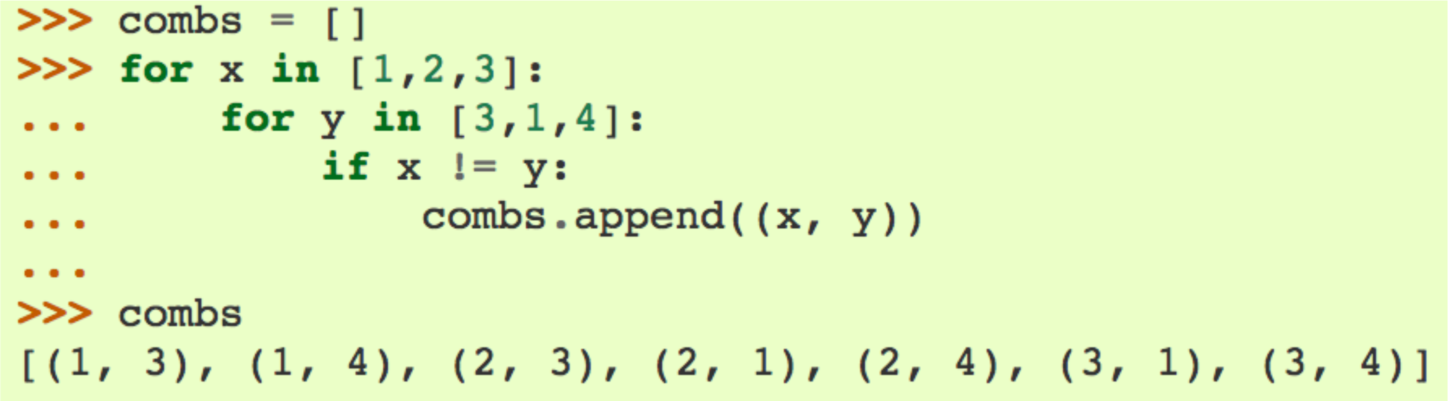


```python
import copy

class Foo(object):
    def __init__(self, val):
         self.val = val

    def __repr__(self):
        return str(self.val)

foo = Foo(1)

a = ['foo', foo]
b = a[:]
c = list(a)
d = copy.copy(a)
e = copy.deepcopy(a)

# edit orignal list and instance
a.append('baz')
foo.val = 5
```

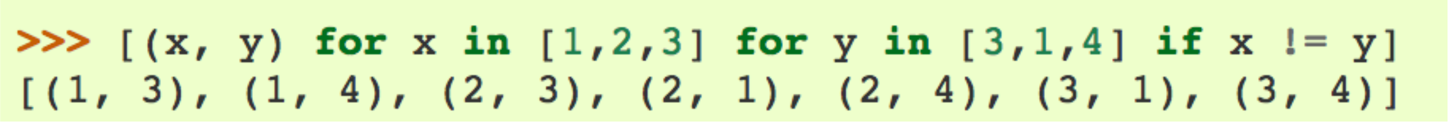
- What would be a,b,c,d,e respectively?

More about Python List

- List Comprehensions



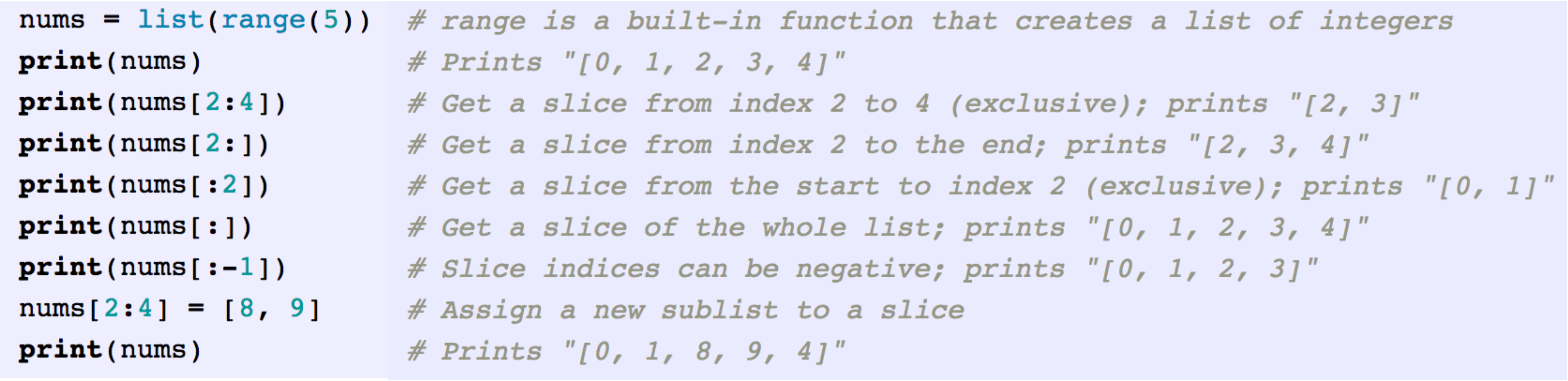
```
>>> combs = []
>>> for x in [1,2,3]:
...     for y in [3,1,4]:
...         if x != y:
...             combs.append((x, y))
...
>>> combs
[(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]
```



```
>>> [(x, y) for x in [1,2,3] for y in [3,1,4] if x != y]
[(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]
```

More about Python List

- Slicing: In addition to accessing list elements one at a time, Python provides concise syntax to access sub lists; this is known as *slicing*. All slicing returns a new copy list:



```
nums = list(range(5))    # range is a built-in function that creates a list of integers
print(nums)              # Prints "[0, 1, 2, 3, 4]"
print(nums[2:4])         # Get a slice from index 2 to 4 (exclusive); prints "[2, 3]"
print(nums[2:])          # Get a slice from index 2 to the end; prints "[2, 3, 4]"
print(nums[:2])          # Get a slice from the start to index 2 (exclusive); prints "[0, 1]"
print(nums[:])           # Get a slice of the whole list; prints "[0, 1, 2, 3, 4]"
print(nums[:-1])         # Slice indices can be negative; prints "[0, 1, 2, 3]"
nums[2:4] = [8, 9]       # Assign a new sublist to a slice
print(nums)              # Prints "[0, 1, 8, 9, 4]"
```

- Numpy tutorial
  - https://github.com/mingrammer/cs231n-numpy-tutorial/blob/master/numpy_tutorial.ipynb

Classes and Objects

- A software item that contains variables and methods
- Object Oriented Design focuses on
  - Encapsulation:
    - dividing the code into a public interface, and a private implementation of that interface
  - Polymorphism:
    - the ability to overload standard operators so that they have appropriate behavior based on their context
  - Inheritance:
    - the ability to create subclasses that contain specializations of their parents

Misc. File I/O, Strings, Exceptions…

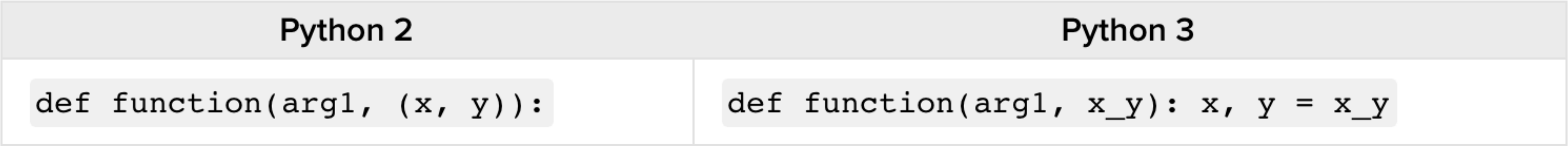
```
>>> try:
...     1 / 0
... except:
...     print('That was silly!')
... finally:
...     print('This gets executed no matter what')
...
That was silly!
This gets executed no matter what
```

```
fileptr = open(‘filename’)
somestring = fileptr.read()
for line in fileptr:
   print line
fileptr.close()
```

```
>>> a = 1
>>> b = 2.4
>>> c = 'Tom'
>>> '%s has %d coins worth a total of $%.02f' % (c, a, b)
'Tom has 1 coins worth a total of $2.40'
```

Python 2.x vs. Python 3.x

- http://nbviewer.jupyter.org/github/rasbt/python_reference/blob/master/tutorials/key_differences_between_python_2_and_3.ipynb
- Division operator (automatic casting)
- print function (**parenthesis** required)
- xrange (removed)
- Error Handling (**as** required)
- _future_ module (transition)

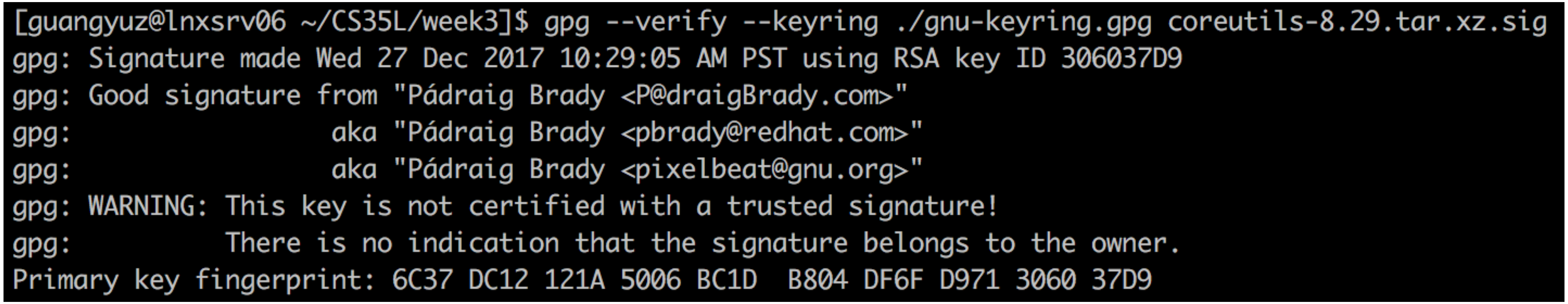


Python 2	Python 3
`def function(arg1, (x, y)):`	`def function(arg1, x_y): x, y = x_y`

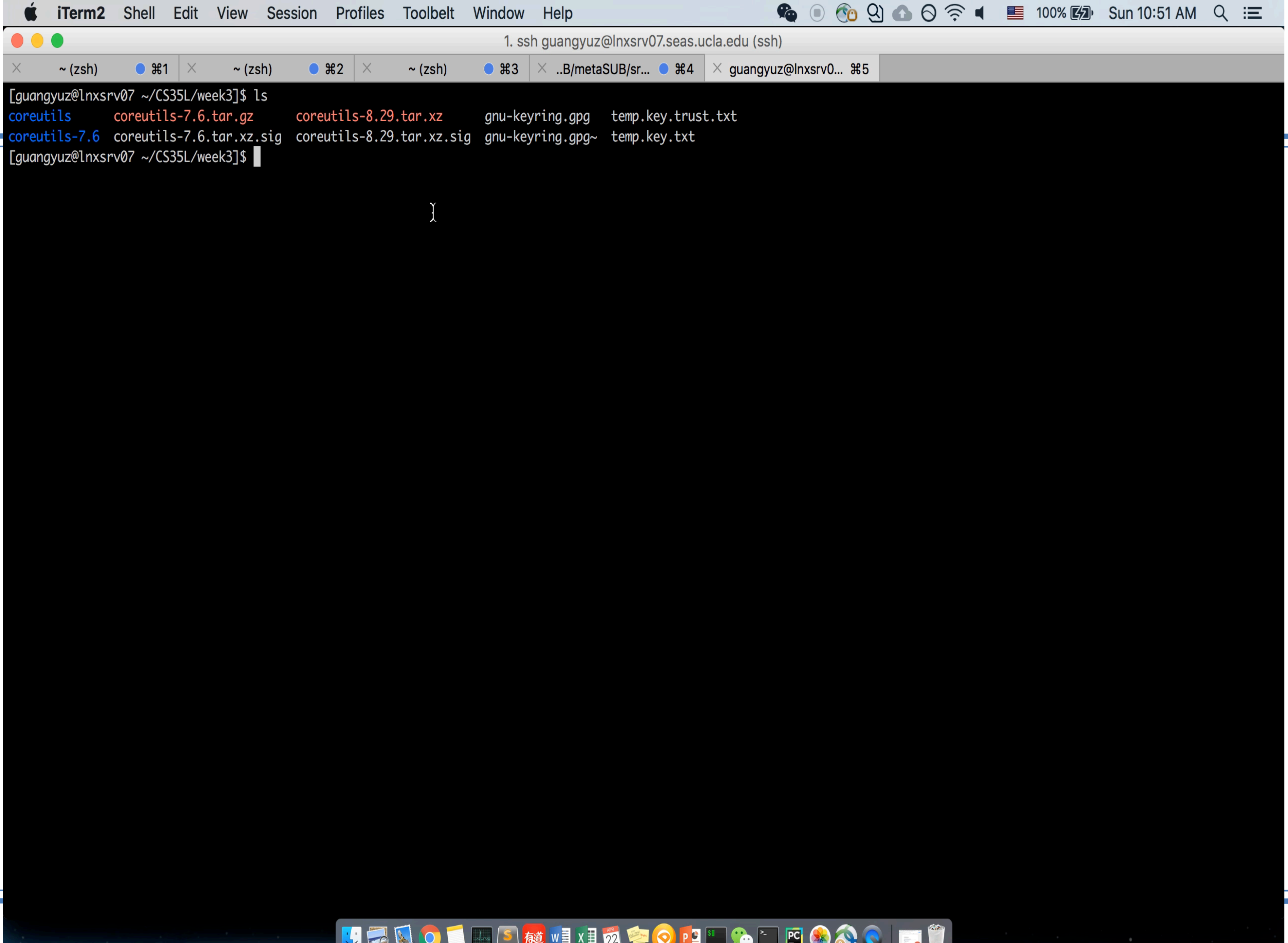
- More …

Lab 3

- You have verified you package.
- Don't worry about the "WARNING: This key is not certified with a trusted signature!" This only means that you have not signed the key with your own key (which you probably don't even have); it's not a message about the safety of the package itself.



```
[guangyuz@lnxsrv06 ~/CS35L/week3]$ gpg --verify --keyring ./gnu-keyring.gpg coreutils-8.29.tar.xz.sig
gpg: Signature made Wed 27 Dec 2017 10:29:05 AM PST using RSA key ID 306037D9
gpg: Good signature from "Pádraig Brady <P@draigBrady.com>"
gpg:                 aka "Pádraig Brady <pbrady@redhat.com>"
gpg:                 aka "Pádraig Brady <pixelbeat@gnu.org>"
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: 6C37 DC12 121A 5006 BC1D  B804 DF6F D971 3060 37D9
```



iTerm2 Shell Edit View Session Profiles Toolbelt Window Help
100% Sun 10:51 AM
1. ssh guangyuz@lnxsrv07.seas.ucla.edu (ssh)
~ (zsh) ⌘1
~ (zsh) ⌘2
~ (zsh) ⌘3
..B/metaSUB/sr... ⌘4
guangyuz@lnxsrv0... ⌘5

```
[guangyuz@lnxsrv07 ~/CS35L/week3]$ ls
coreutils      coreutils-7.6.tar.gz      coreutils-8.29.tar.xz      gnu-keyring.gpg   temp.key.trust.txt
coreutils-7.6  coreutils-7.6.tar.xz.sig  coreutils-8.29.tar.xz.sig  gnu-keyring.gpg~  temp.key.txt
[guangyuz@lnxsrv07 ~/CS35L/week3]$ 
```

Remind: Lab Fixing a bug

- For these users the command la -A is therefore equivalent to ls -a -A.
- Unfortunately, with Coreutils ls, the -a option always overrides the -A option regardless of which option is given first, so the -A option has no effect in la.
- For example, 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. That is, for ls they want a later -A option to override any earlier -aoption, and vice versa.

Lab: Installing a small change to a big package

- Download the tar file of coreutils

  wget [url]

- Extract files

  tar -xzvf

- Compile the file
  - ./configure --prefix=[your home directory]/coreutils
  - **Hint: use absolute path here!**
  - make
  - make install

x means extract files from the archive.
z means (un)zip.
v means print the filenames verbosely.
f means the following argument is a filename.

Lab: Installing a small change to a big package

- Reproduce the bug
  - Export the locale

    export LC_ALL='en_US.UTF-8‘
  - Go to the /bin directory
  - Run **./ls -aA /bin/bash**, don’t use **ls –aA /bin/bash**

Lab: Installing a small change to a big package

- Apply the patch
  - Create the .diff file

    copy and paste from Brady’s patch
  - Use patch command, where you need to specify n

    patch –p[n] > [diff file]
  - Specify the file to be patched

    ls.c

Lab: Installing a small change to a big package

- Recompile and Check
  - Recompile: cd .. make

    DO NOT make clean!
  - Check: go to parent directory
    - Unmodified

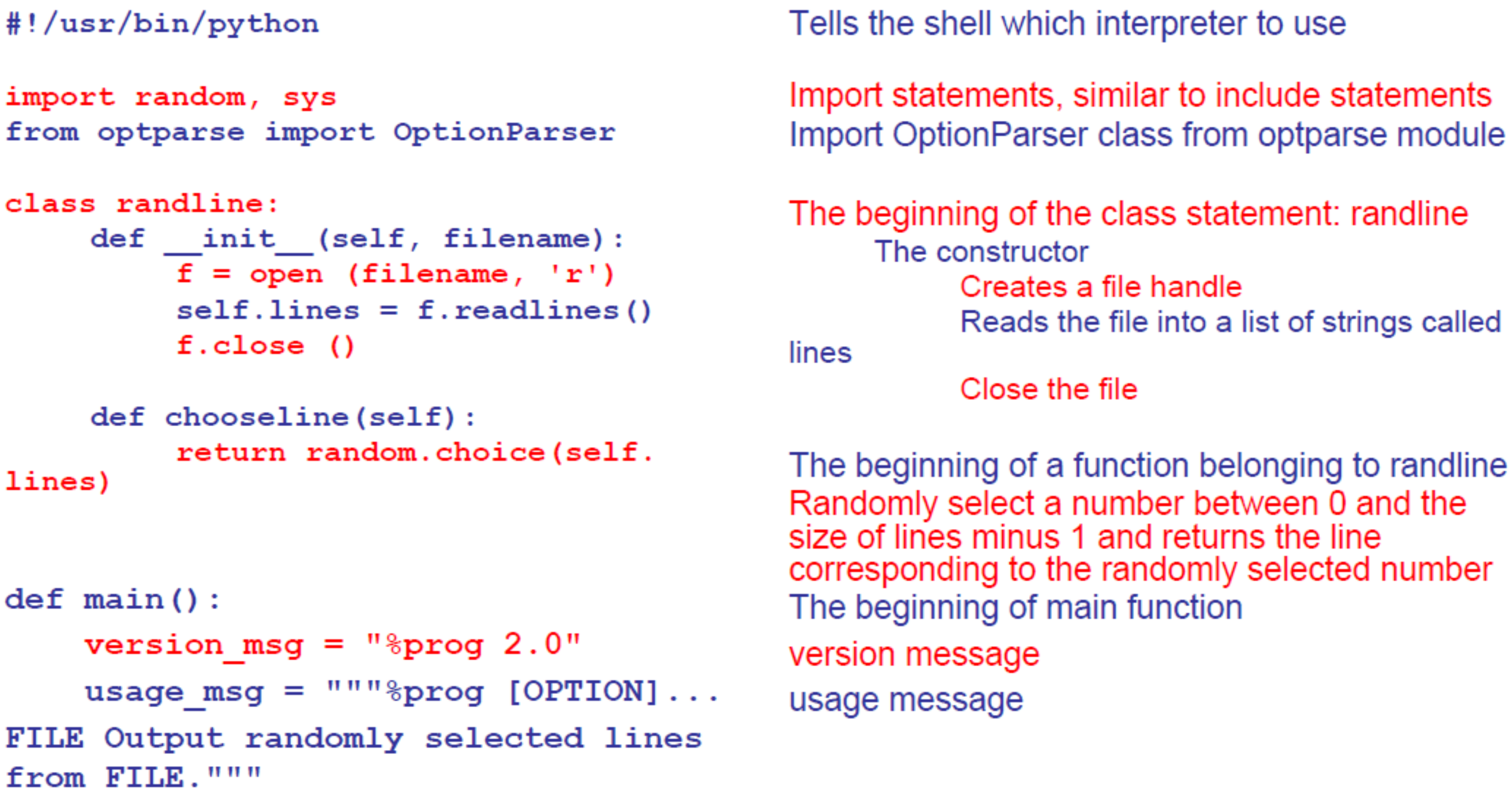
      ./coreutils/bin/ls -aA ./coreutils-8.29.tar.gz
    - Modified

      ./coreutils-8.29/src/ls -aA ./coreutils-8.29.tar.gz
- Test a file that is at least one year old
  - Hints: use command: touch -t

Homework: rewrite a script

- The randline.py program
  - Input: a file and a number *n*
  - Output: *n* random lines from *file*
  - Get familiar with language + understand the program
  - Answer some questions about script
- Port a program to python 3
  - /usr/local/cs/bin/python3 randline.py /dev/null
- To run Python3+:
  - export PATH=/usr/local/cs/bin/:$PATH
  - python3

randline.py 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 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 minus 1 and returns the line corresponding to the randomly selected number
The beginning of main function
version message
usage message

randline.py 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

Implement the shuf command: C -> python

- shuf:
  - Write a random permutation of the input lines to standard output
- Support the following shuf options, with the same behavior as GNU shuf:

  --input-range (-i), --head-count (-n), --repeat (-r), and --help.
- Also support any number (including zero) of non-option arguments, as well as the argument "-" meaning standard input.
- Change usage message to describe script behavior
- Port shuf.py to Python 3
- Follow the instruction on Piazza

Homework 3 Hints

- Q4: Python 3 vs. Python 2
  - Look up “automatic tuple unpacking”
- Check the shuf utility source
  - Use the same logic
- Run "shuf --version" to test compatibility with using Coreutils 8.29
  - Installed in /usr/local/cs/bin on lnxsrv06, 07, 09, 10)
- Remember to support input from STDIN
  - $ cat input1.txt | python shuf.py –
- Use randline.py as a starting point
  - Modify to work exactly like shuf