CS 35L- Software Construction Laboratory

Fall 18

TA: Guangyu Zhou

Lab 3

Reminder

- Signup for Assignment 10 Presentation (No later than Oct 21, 11:55pm)
  - Use **UCLA** account to register at the following link
  - https://docs.google.com/spreadsheets/d/1L2IeP7WRbCmAygSPN5pq7SU2pTddKNooUteBGetde2M/edit?usp=sharing
  - Topic on recent research in computer science
    - **Technical** content is required
  - 1 or 2 people
  - ~10 minutes talk in class (~12 min for teams)
  - Use slides and upload to CCLE before presentation
  - Participation in Q&A
  - Brief Research report (due in the last week)
- Purchase the Beagle Bones as early as possible

Course Information

- Assignment 2 due Today
- Assignment 10 presentation
  - **Specify your topic ASAP if haven't, or I will clear your slots.**
  - 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.**

Outline

- Build from source & Bug Fixing
- Compile using makefile
- **Introduction to Python**

Review: Build Process

- **configure**
  - Script that checks details about the machine before installation
    - Dependency between packages
  - 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)

Review: 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
- Patch Command
  - Usage: patch [options] [originalfile] [patchfile]
  - **-pnum**: strip the smallest prefix containing num leading slashes from each file name found in the patch file
  - Examples: see supplement materials [Patch command]

Introduction to Python 2.x

- Not just a scripting language, open source general-purpose language
- Object-Oriented language
  - Support Class
  - Support member functions
- Compiled and interpreted
  - Python code is compiled to bytecode
  - Bytecode interpreted by Python interpreter
- Not as efficient as C, but easy to learn, read and use
- Easy to interface with C/ObjC/Java/Fortran
- Great interactive environment

Python Interpreter

- Interactive interface to Python

```
% python
Python 2.5 (r25:51908, May 25 2007, 16:14:04)
[GCC 4.1.2 20061115 (prerelease) (SUSE Linux)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

- Python interpreter evaluates inputs:

```
>>> 3*(7+2)
27
```

- Python prompts with ‘>>>’.
- To exit Python: CTRL-D

Sample of Python code

```
x = 34 - 23              # A comment.
y = “Hello”              # Another one.
z = 3.45
if z == 3.45 or y == “Hello”:
    x = x + 1
    y = y + “ World”     # String concat.
print x
print y
```

- **Assignment uses = and comparison uses ==.**
- **For numbers + - * / % are as expected.**
  - Special use of + for string concatenation.
  - Special use of % for string formatting (as with printf in C)
- **Logical operators are words (`and, or, not`), *not* symbols**
- **The basic printing command is `print`.**
- **The first assignment to a variable creates it.**
  - Variable types don’t need to be declared.
  - Python figures out the variable types on its own.

Basic data types

- **Integers (default for numbers)**
  - `z = 5 / 2     # Answer is 2`
- **Floats**
  - `x = 3.456`
- **Strings**
  - Can use “” or ‘’ to specify.
  - `“abc”  ‘abc’` (Same thing.)
  - Unmatched can occur within the string.
  - `“matt’s”`
  - Use triple double-quotes for multi-line strings or strings than contain both ‘ and “ inside of them:
  - `“““a‘b“c”””`

Python indentation

- Whitespace is meaningful in Python: especially indentation and placement of newlines.
- Use a newline to end a line of code.
  - Use \ when must go to next line prematurely.
- No braces { } to mark blocks of code in Python… Use consistent indentation instead.
  - The first line with less indentation is outside of the block.
  - The first line with more indentation starts a nested block
- Often a colon appears at the start of a new block. (E.g. for function and class definitions.)

Python variable & assignment

- Binding a variable in Python means setting a name to hold a reference to some object.
  - Assignment creates references, not copies
- Names in Python do not have an intrinsic type. Objects have types.
  - Python determines the type of the reference automatically based on the data object assigned to it.
- You create a name the first time it appears on the left side of an assignment expression:
  - x = 3
- A reference is deleted via garbage collection after any names bound to it have passed out of scope.

Python variable & assignment

You can also assign to multiple names at the same time.

```
>>> x, y = 2, 3
>>> x  2
>>> y  3
```

Naming rules

- Names are case sensitive and cannot start with a number. They can contain letters, numbers, and underscores.

  bob Bob _bob _2_bob_ bob_2 BoB
- There are some reserved words:

  and, assert, break, class, continue, def, del, elif, else, except, exec, finally, for, from, global, if, import, in, is, lambda, not, or, pass, print, raise, return, try, while

Python sequence types

- Tuple
  - A simple **immutable** ordered sequence of items
  - Items can be of mixed types, including collection types
- String
  - **Immutable**
  - Conceptually very much like a tuple
- List:
  - **Mutable** ordered sequence of items of mixed types

Sequence type examples

- **Tuples are defined using parentheses (and commas).**
- `>>>tu = (23, ‘abc’, 4.56, (2,3), ‘def’)`
- **Lists are defined using square brackets (and commas).**
- `>>>li = [“abc”, 34, 4.34, 23]`
- **Strings are defined using quotes (“, ‘, or “““).**
- `>>>st = “Hello World”`
- `>>>st = ‘Hello World’`
- `>>>st = “““This is a multi-line  string that`

```
uses triple quotes.”””
```

Sequence type examples, cont’d

```
>>> tu = (23, ‘abc’, 4.56, (2,3), ‘def’)
>>> tu[1]     # Second item in the tuple.
 ‘abc’

>>> li = [“abc”, 34, 4.34, 23]
>>> li[1]      # Second item in the list.
 34

>>> st = “Hello World”
>>> st[1]   # Second character in string.
 ‘e’
```

Mutability: Tuples vs. Lists

- Tuples: immutable
- `>>> t = (23, ‘abc’, 4.56, (2,3), ‘def’)`
- `>>> t[2] = 3.14`
- `Traceback (most recent call last):`
  - `File "<pyshell#75>", line 1, in -toplevel-`
    `tu[2] = 3.14`
- `TypeError: object doesn't support item assignment`
- **You can’t change a tuple.**
- **You can make a fresh tuple and assign its reference to a previously used name.**
- `>>> t = (23, ‘abc’, 3.14, (2,3), ‘def’)`

Mutability: Tuples vs. Lists

- List: Mutable
- `>>>li = [‘abc’, 23, 4.34, 23]`
- `>>>li[1] = 45`
- `>>>li`
  - `[‘abc’, 45, 4.34, 23]`

- **We can change lists *in place.***
- **Name *li* still points to the same memory reference when we’re done.**
- **The mutability of lists means that they aren’t as fast as tuples.**

Tuples vs. Lists

- Lists slower but more powerful than tuples.
  - Lists can be modified, and they have lots of handy operations we can perform on them.
  - Tuples are immutable and have fewer features.

- To convert between tuples and lists use the list() and tuple() functions:
  - `li = list(tu)`
  - `tu = tuple(li)`

Lists operations

```
>>> li = [1, 11, 3, 4, 5]

>>> li.append(‘a’)   # Our first exposure to method syntax
>>> li
[1, 11, 3, 4, 5, ‘a’]

>>> li.insert(2, ‘i’)
>>>li
[1, 11, ‘i’, 3, 4, 5, ‘a’]

>>> li.extend([9, 8, 7])
>>>li
[1, 2, ‘i’, 3, 4, 5, ‘a’, 9, 8, 7]
```

+ vs extend:

+ creates a fresh list (with a new memory reference)
***extend* operates on list `li` in place.**

Lists operations, cont’d

```
>>> li = [‘a’, ‘b’, ‘c’, ‘b’]

>>> li.index(‘b’)     # index of first occurrence
1

>>> li.count(‘b’)     # number of occurrences
2

>>> li.remove(‘b’)    # remove first occurrence
>>> li
  [‘a’, ‘c’, ‘b’]
```

```
>>> li = [5, 2, 6, 8]

>>> li.reverse()    # reverse the list *in place*
>>> li
  [8, 6, 2, 5]

>>> li.sort()       # sort the list *in place*
>>> li
  [2, 5, 6, 8]

>>> li.sort(some_function)
    # sort in place using user-defined comparison
```

Python Dictionary

- Essentially a hash table
  - Provides key-value (pair) storage capability

- You can define, modify, view, lookup, and delete the key-value pairs in the dictionary.

- Instantiation:
  - dict = {}
  - This creates an EMPTY dictionary

- Keys are unique, values are not!
  - Keys must be immutable (strings, numbers, tuples)

Example

```
>>> d = {‘user’:‘bozo’, ‘pswd’:1234}
>>> d[‘user’]
‘bozo’
>>> d[‘pswd’]
1234
>>> d[‘bozo’]

Traceback (innermost last):
  File ‘<interactive input>’ line 1, in ?
KeyError: bozo

>>> d = {‘user’:‘bozo’, ‘pswd’:1234}
>>> d[‘user’] = ‘clown’
>>> d
{‘user’:‘clown’, ‘pswd’:1234}

>>> d[‘id’] = 45
>>> d
{‘user’:‘clown’, ‘id’:45, ‘pswd’:1234}
```

```
>>> d = {‘user’:‘bozo’, ‘p’:1234, ‘i’:34}
>>> del d[‘user’]           # Remove one.
>>> d
{‘p’:1234, ‘i’:34}
>>> d.clear()               # Remove all.
>>> d
{}

>>> d = {‘user’:‘bozo’, ‘p’:1234, ‘i’:34}
>>> d.keys()              # List of keys.
[‘user’, ‘p’, ‘i’]
>>> d.values()            # List of values.
[‘bozo’, 1234, 34]
>>> d.items()        # List of item tuples.
[(‘user’,‘bozo’), (‘p’,1234), (‘i’,34)]
```

Python control flows

```
if x == 3:
    print "X equals 3."
elif x == 2:
    print "X equals 2."
else:
    print "X equals something else."
print "This is outside the 'if'."
```

```
assert(number_of_players < 5)
```

```
x = 3
while x < 10:
   if x > 7:
       x += 2
       continue
   x = x + 1
   print "Still in the loop."
   if x == 8:
       break
print "Outside of the loop."
```

```
for x in range(10):
   if x > 7:
       x += 2
       continue
   x = x + 1
   print "Still in the loop."
   if x == 8:
       break
print "Outside of the loop."
```

Python functions

- def creates a function and assigns it a name
- return sends a result back to the caller
- Arguments are passed by assignment
- Arguments and return types are not declared

```
def <name>(arg1, arg2, ..., argN):
  <statements>
  return <value>

def times(x,y):
  return x*y
```

Python modules

- Code reuse
  - Routines can be called multiple times within a program
  - Routines can be used from multiple programs
- Namespace partitioning
  - Group data together with functions used for that data
- Implementing shared services or data
  - Can provide global data structure that is accessed by multiple subprograms

Python modules, cont'd

- Modules are functions and variables defined in separate files
- Items are imported using from or import

  ```
  from module import function
  function()
  ```
  Or:
  ```
  import module
  module.function()
  ```
- Modules are namespaces
  - Can be used to organize variable names, i.e.
  - `atom.position = atom.position - molecule.position`

Optparse library/module

- 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. It is consumed from the argument list when the option is defined

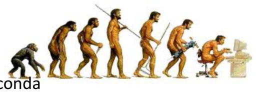
Running Python scripts

- Check *example.py* file in supplement materials
- Create a same file with same name on Seasnet server
- Assign executable permission

  chmod +x
- Run it
  - python ./example.py

A more powerful Environment

- Higher mammals use advanced tools !



- Anaconda
  - An data science platform powered by python
  - Support Different OS(Windows, Mac OS, Linux)
  - Easy to use
  - Powerful Tools (Jupyter notebook)

  https://www.continuum.io/downloads

Supplement resources

- Python Tutorial
  - https://docs.python.org/3.5/tutorial/
- Python Examples
  - http://www.programiz.com/python-programming/examples
- Anaconda
  - https://www.continuum.io/downloads
- Demo of Jupyter Notebook
  - http://nbviewer.jupyter.org/github/jdwittenauer/ipython-notebooks/blob/master/notebooks/language/Intro.ipynb

Task: 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 -l /bin/bash**, don’t use **ls –l /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