# Modify and Rewrite Programs

Week 3

# Make

make [OPTION]... [TARGET]...

- GNU utilities to maintain groups of program
- Automatically determine which part of large program needs to be recompiled
  - Make update a target if it depends on prerequisite files that have been modified since the target was last modified, or if the target does not exist
- Efficient compilation
- Take a Makefile to specify all target and prerequisite

# Makefile Example

```
# Makefile - A Basic Example
all : shop  #usually first
shop : item.o shoppingList.o shop.o
        g++ -g -Wall -o shop item.o shoppingList.o shop.o
item.o : item.cpp item.h
        g++ -g -Wall -c item.cpp
shoppingList.o : shoppingList.cpp shoppingList.h
        g++ -g -Wall -c shoppingList.cpp
shop.o : shop.cpp item.h shoppingList.h
        g++ -g -Wall -c shop.cpp
clean :
        rm -f item.o shoppingList.o shop.o shop
```

Rule

Comments

Targets

Prerequisites

Dependency Line

Commands

# Introduction to Python

- High-level

  (high readability, not efficient as C)
- General-purpose
- Interpreted
- Dynamic (dynamic type system)
- Automatic memory management
- Also support Object-oriented programming
  - Support class
  - Support member function

# 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
- 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]**

# Python Dictionary

- Essentially a hash table
  - Provides key-value (pair) storage capability
- Instantiation:
  - dict = {}
  - This creates an EMPTY dictionary
- Keys are unique, values are not!
  - Keys must be immutable (strings, numbers, tuples)

# Example

- dict = {}
- dict[‘hello’] = “world”
- print dict[‘hello’]
  - world
- dict[‘power’] = 9001
- if (dict[‘power’] > 9000):
- print “It is over “, dict[‘power’]
  - It is over 9001
- del dict[‘hello’]
- del dict

# 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:**

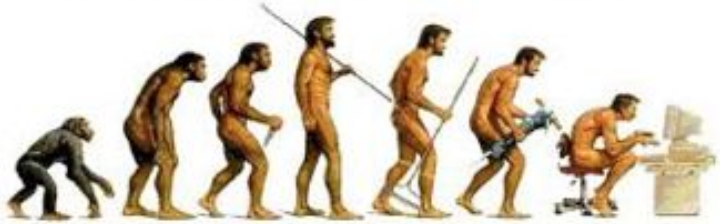
0
1
2
3
4

# 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!!

# 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

# Running Python scripts

- Use the example in randlin.py
- Make sure it has executable permission:

  chmod +x randline.py

- Run it

  ./randline.py –n 2 filename

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

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

  Filename: is a program argument

```
#!/usr/bin/python
```

Tells the shell which interpreter to use

```
import random, sys
from optparse import OptionParser
```

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

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

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

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 and returns the line corresponding to the randomly selected number

```
def main():
    version_msg = "%prog 2.0"
    usage_msg = """%prog [OPTION]...
FILE Output randomly selected lines from
FILE."""
```

The beginning of main function

```
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.chooselin
e())
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

# Homework 3

comm.py

# Homework 3

- comm.py – this should end up working almost exactly like the utility 'comm'
  - Check $ man comm for extensive documentation
- Extra option –u
  - Means input files are not required to be pre-sorted
  - Could sort them, but then have to maintain original ordering
    - Other ways to accomplish this?

# Comm.py

- Use randline.py as a start point
- Support all options for comm

–-1, -2, -3 and combinations

–Extra option –u for comparing unsorted files

- Support all type of arguments

–File names and – for stdin

- If you are unsure of how something should be output, run a test using existing comm utility!

–Create your own test inputs

# Reference

- Optparse tutorial

https://docs.python.org/2/library/optparse.html

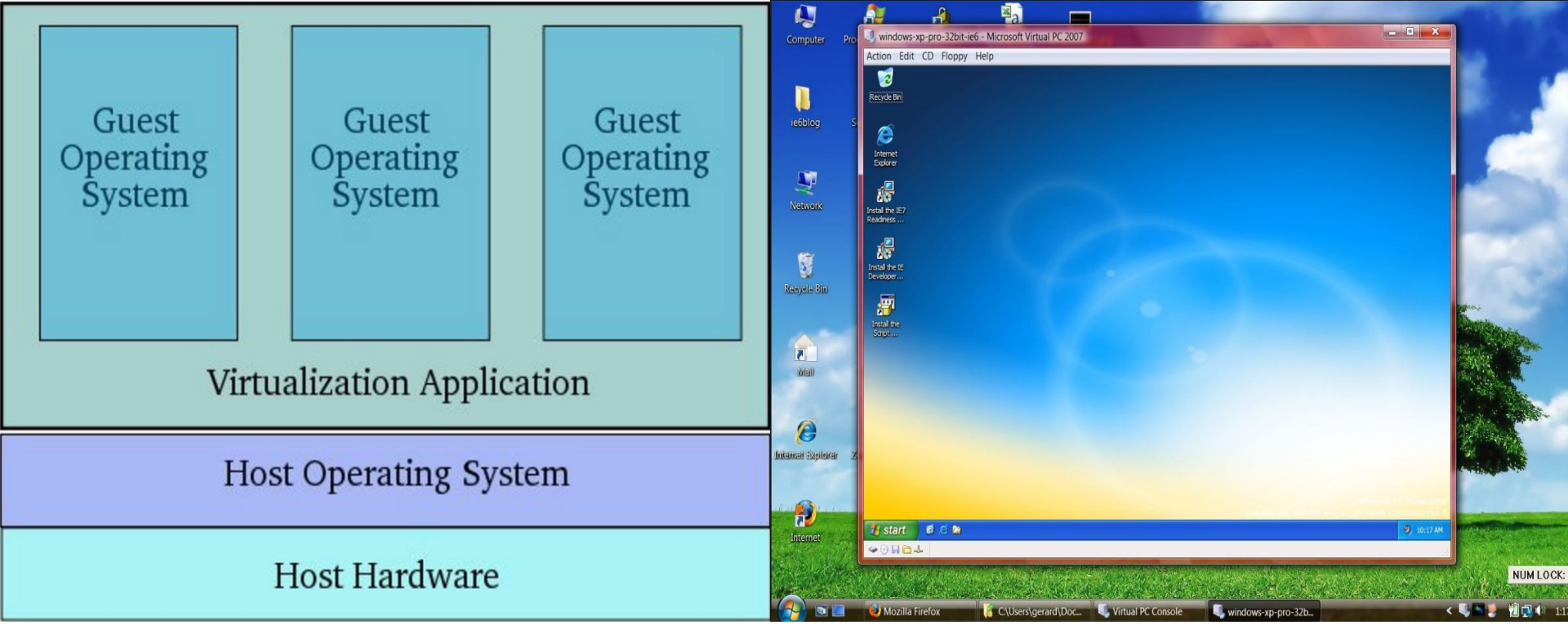
- Python tutorial

https://docs.python.org/3/tutorial/

# Virtual Machine

- A program that acts like a virtual computer
- Runs on host OS and provides virtual hardware to guest OS
- Guest OS runs in a window on host OS, just like any other program
- From user's perspective, guest OS seems to be running on a physical machine
- Some popular virtual machine software:

  VirtualBox, VMvare




Some applications

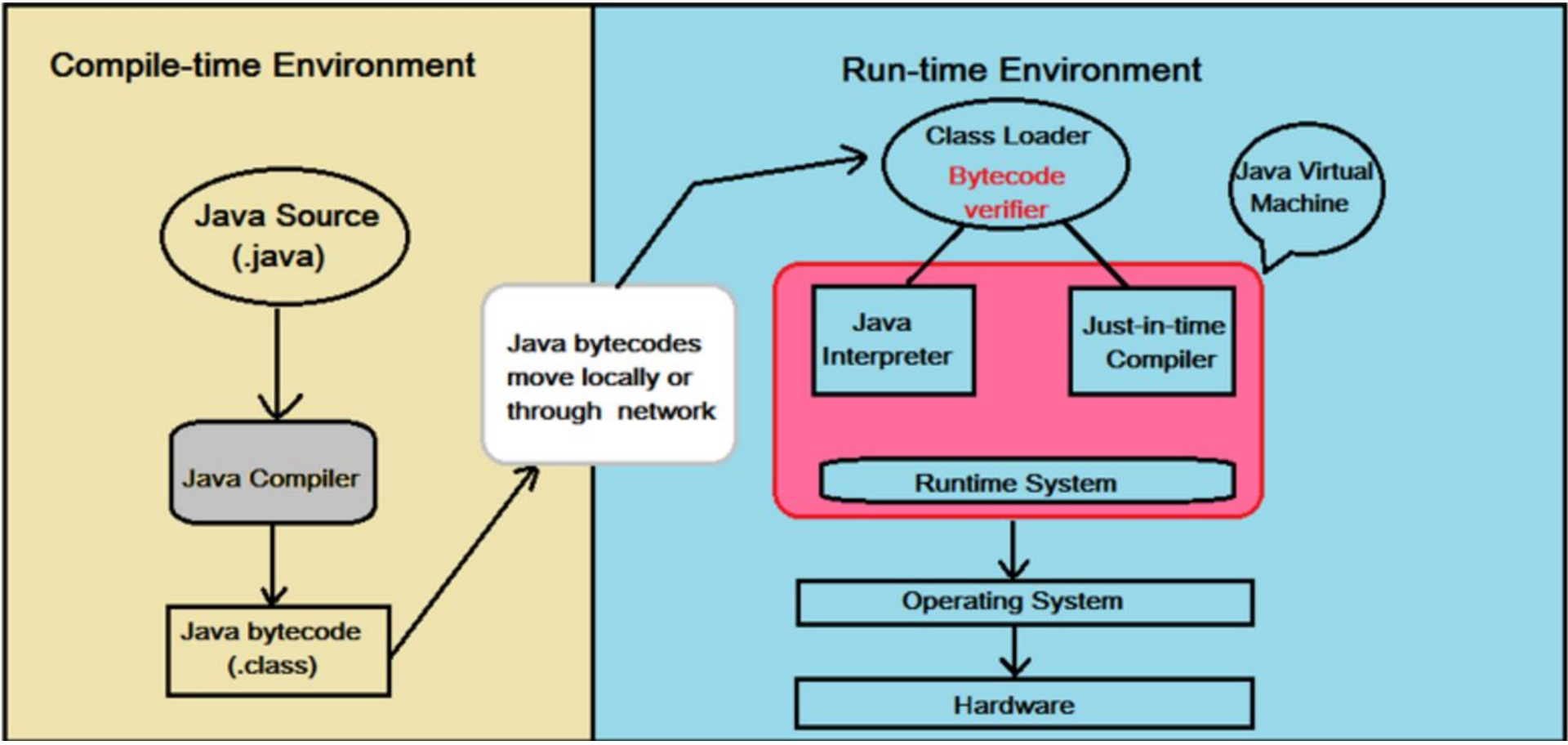
- Experiment with other OS
- Test software on multiple platforms
- Consolidate servers

# Java virtual machine (JVM)

- An abstract computing machine that enables a computer to run a java program
- JVM is OS specific
- JVM interprets .class (bytecode) file and converts it to machine specific instruction set
- JVM provides a platform independent way of executing code, which makes java very portable

# Two-step compilation process

- First stage: java code is compiled down to bytecode by java compiler (javac)
- Second stage: bytecode is then interpreted or compiled to run by JVM depending on the implementation of JVM


Compile-time Environment
Java Source
(.java)
Java Compiler
Java bytecode
(.class)
Java bytecodes
move locally or
through network
Run-time Environment
Class Loader
Bytecode
verifier
Java Virtual
Machine
Java
Interpreter
Just-in-time
Compiler
Runtime System
Operating System
Hardware

# Java basics

- Java as a compromise between interpreted and compiled language
- JRE (Java runtime environment)

  JVM + Java class libraries
- JDK (Java Development Kit)

  JVM + Java class libraries + java compiler