LSPU Self-paced Learning Module (SLM)

Course	**Bachelor of Science in Computer Science**
Sem/AY	**First Semester/2023-2024**
Module No.	**3**
Lesson Title	**Object-oriented Programming, Scripting Languages**
Week Duration	**4**
Date	
Description of the Lesson	This lesson will discuss the fundamental features and concepts to different programming languages. Topics include overview of programming languages, Introduction to language translation, type systems, data and execution control, declaration and modularity, and syntax and semantics. Laboratory will be used to demonstrate each of the concepts using different programming languages.

Learning Outcomes

Intended Learning Outcomes	Students should be able to meet the following intended learning outcomes: • Describe the basic objects and constructs in Object-Oriented Programming. • Explain the functional groups in object-oriented programming languages. • Classify appropriate functional languages into an appropriate commercial or academic field. • Demonstrate how to evaluate an expression utilizing Lambda-Calculus. • Apply higher-order procedures to an application set. • Explain the characteristics of pure functional functions in functional programming.
Targets/ Objectives	At the end of the lesson, students should be able to learn, practice and apply the ff: • *Fundamental Features of Programming* • *Object Types and Typing* • *Java Virtual Machine* • *Templates and Generics* • *Scripting Language* • *Regular Expressions* • *Ruby* • *Python* • *JavaScript*

Student Learning Strategies

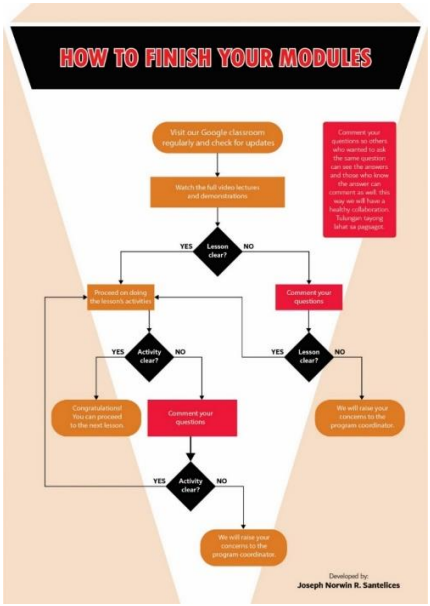
Online Activities (Synchronous/ Asynchronous)	

A. Online Discussion via Google Classroom.
You will be directed to join in the online classroom where you will have access to the video lectures, course materials, and evaluation tools 24/7. To have access on the Online Lectures, refer to this link: _____.

To ensure a healthy collaboration, you may post your inquiries on a particular topic and the instructor including all the members of the class can acknowledge the inquiry.

(For further instructions, refer to your Google Classroom and see the schedule of activities for this module)

B. Learning Guide Questions:
Refer to the following diagram as your guide through the course.



Note: *The insight that you will post on online discussion forum using Learning Management System (LMS) will receive additional scores in class participation*

Offline Activities (e-Learning/Self-Paced)

Lecture Guide

Generalizations about Scripting Languages

Interpreted (some compiled internally)

Rapid Development — no compilation required

Portability — cross platform (since not compiling — no machine code)

Slower — typically slower than compiled languages… oh well

Frequently no “main”

Good for controlling other applications

Flexibility

Easy to extend

Vast libraries and extensions available

Popular Applications of Scripting Languages

System Administration

System Automation

Text Manipulation

Web Development

Client-side

Server-side

Graphical User Interfaces

System Prototyping

Hacking & Cracking

Gluing

Interfacing Incompatible Programs

Connecting Programs

What are Regular Expressions?

Regular expressions are patterns of characters that match, or fail to match, sequences of characters in text. Regular expressions have their own syntax where certain characters and combinations of characters have special meanings and uses.

Abbreviated as “regexp”, “regexq”, or simply “re”
Pattern consists of literals and meta-characters
Typically implemented using either a finite state machine or via backtracking

Examples:

```
1   /^[A-E]/                    # match all lines that begin A
2   /\s*rem/i                   # match DOS shell comments
3   /^\S{1,8}\.\S{0,3}/         # match DOS 8.3 style file name
```

Applications of Regular Expressions

Finding doubled words
the quick brown fox fox jumps over the the lazy dog → the quick brown fox jumps over the lazy dog
Validating input
Web based forms
Traditional GUI forms
Changing formats
Dates: 5/16/78 → 05-16-1978
Phone numbers: 123-456-7890 → (123) 456-7890
Fixing case issues
the latest version of the Javascript language is javascript 1.7 → the latest version of the JavaScript language is JavaScript 1.7
HTML-ifying documents
Visit http://www.w3.org/ for more information → Visit http://www.w3.org/ for more information

Regular Expressions You Already Use

Search and replace in word processors

Dealing with files

rm *.pl

ls ??.html

emacs *.[ch]

rm -rf *.{o,bak}

Searching online

Why Regular Expressions Seem Intimidating

Cryptic

Compact

Whitespace sensitive (typically)

No standards body

Differences between implementations

Character changes meaning based upon context

Often case sensitive

Evolution of regular expression engines

Multiple solutions for a given problem

Often takes time to fine tune the regular expression — tends to be an iterative process

Support for Regular Expressions

Scripting Languages

Perl

JavaScript

PHP (supports own native regex engine as well as Perl's)

Python

Tcl

Visual Basic

Ruby

Many others...

Systems Languages

C/C++

C#

Java

UNIX tools

grep

awk

sed

Various shells (bash, csh, tcsh, sh, etc...)

Editors

emacs/xemacs

vi

MS Office

Open Office

Many other editors...

Other Specialized Tools

expect

procmail

Are Regular Expressions Still Relevant?

One might wonder if regular expressions are still relevant in the XML age — the short answer is yes.

There are many systems out there which predate the acceptance of XML as a data interchange standard. Data from these systems cannot be be parsed with an XML parser and may be parsed directly by utilizing regular expressions. Alternatively, the data could even be converted to XML (by use of regular expressions) for more “modern” processing using various XML tools.

Even well formatted XML data (which can be easily parsed) can benefit from use of regular expressions. For example, XQuery and XPath both have support for regular expressions.

Our Study

We will start with an introductory look at syntax and usage of Regular Expressions

We will revisit them in more detail in the various languages that we examine

For now, focus on the regular expression not the language

What is Ruby?

Ruby is an interpreted scripting language for quick and easy object-oriented programming. It has many features to process text files and to do system management tasks (as in Perl). It is simple, straight-forward, and extensible.

If you want a language for easy object-oriented programming, or you don' like the Perl ugliness, or you do like the concept of LISP, but don't like too much parentheses, Ruby may be the language of your choice.

Ruby originated in Japan in the mid 1990s by Yukihiro “Matz” Matsumoto.

Ruby History

Late 1995

Ruby 0.95 release was announced on Japanese newsgroups

Late 1996

Ruby 1.0 released

1999

Ruby 1.3 released, first English language mailing list setup

Late 2000

First English language book on Ruby published — since freely released

Mid 2004

David Heinemeier Hansson first released the now popular Ruby on Rails framework.

January 30th 2009

Ruby 1.9.1 released — first stable release of the Ruby 1.9 series. First release to switch over from Matz's Ruby Interpreter (MRI) to the Yet another Ruby VM (YARV) which is reported to speed up performance 2 to 4 fold.

Today (as of February 7th 2012)

Ruby 1.9.3-p0 is the current stable release. Ruby 2.0 is under active development.

Ruby Features

An overview of Ruby's features, per its manpage:

Interpretive

Ruby is an interpreted language, so you don't have to recompile programs written in Ruby to execute them.

Variables have no type (dynamic typing)

Variables in Ruby can contain data of any type. You don't have to worry about variable typing. Consequently, it has a weaker compile time check.

No declaration needed

You can use variables in your Ruby programs without any declarations. Variable names denote their scope, local, global, instance, etc.

Simple syntax

Ruby has a simple syntax influenced slightly from Eiffel.

No user-level memory management

Ruby has automatic memory management. Objects no longer referenced from anywhere are automatically collected by the garbage collector built into the interpreter.

Everything is an object

Ruby is the purely object-oriented language, and was so since its creation. Even such basic data as integers are seen as objects.

Class, inheritance, and methods

Of course, as an object-oriented language, Ruby has such basic features like classes, inheritance, and methods.

Singleton methods

Ruby has the ability to define methods for certain objects. For example, you can define a press-button action for certain widget by defining a singleton method for the button. Or, you can make up your own prototype based object system using singleton methods, if you want to.

Mix-in by modules

Ruby intentionally does not have the multiple inheritance as it is a source of confusion. Instead, Ruby has the ability to share implementations across the inheritance tree. This is often called “Mix-in”.

Iterators

Ruby has iterators for loop abstraction.

Closures

In Ruby, you can objectify the procedure.

Text processing and regular expression

Ruby has a bunch of text processing features like in Perl.

Bignums

With built-in bignums, you can for example calculate factorial(400).

Exception handling

As in Java.

Direct access to the OS

Ruby can use most UNIX system calls, often used in system programming.

Dynamic loading

On most UNIX systems, you can load object files into the Ruby interpreter on-the-fly.

irb — Interactive Ruby Interpreter

Ruby ships with a program called `irb` — this is Ruby's interactive interpreter. `irb` is great for trying out things in ruby if you want to see how they work without writing a full blown

script to execute.

```
linux3[1]% irb
irb(main):001:0> puts "Hello World"
Hello World
=> nil
irb(main):002:0> 9.18 / 5.16
=> 1.77906976744186
irb(main):003:0> exit
linux3[2]%
```

If you want to simplify that prompt in `irb`, you can execute it with the `--simple-prompt` flag (or alias it in your shell).

```
linux3[2]% irb --simple-prompt
>> puts "foo bar baz"
foo bar baz
=> nil
>> exit
linux3[3]%
```

ri — Ruby Interactive (Information)

Though many of you will probably use the online docs at ruby-doc.org, you can also get at all of the class-level documentation from the command line.

Ruby ships with a program called `ri` which allows you to easily get access to documentation for the built-in classes. You can specify the name of a class to get information about the class as a whole including methods:

```
linux3[1]% ri File

------------------------------------------------------------ Class: File

```

```
5       FTOOLS.RB: EXTRA TOOLS FOR THE FILE CLASS
6       ========================================
7            Author:        WATANABE, Hirofumi
8
9            Documentation: Zachary Landau
10
11           This library can be distributed under the terms of the Ruby
12           license. You can freely distribute/modify this library.
13
14           It is included in the Ruby standard library.
15
16
17      Description
18      -----------
19           ftools adds several (class, not instance) methods to the File
20           class, for copying, moving, deleting, installing, and comparing
21           files, as well as creating a directory path. See the File class for
22           details.
23
24           FileUtils contains all or nearly all the same functionality and
25           more, and is a recommended option over ftools
26
27           When you
28
29             require 'ftools'
30
31           then the File class aquires some utility methods for copying,
32           moving, and deleting files, and more.
33
34           See the method descriptions below, and consider using FileUtils as
35           it is more comprehensive.
36
37      --------------------------------------------------------------------------
38
```

```
39
40      Constants:
41      ----------
42           Separator:      separator
43           SEPARATOR:      separator
44           ALT_SEPARATOR:  rb_obj_freeze(rb_str_new2("\\"))
45           ALT_SEPARATOR:  Qnil
46           PATH_SEPARATOR: rb_obj_freeze(rb_str_new2(PATH_SEP))
47           BUFSIZE:        8 * 1024
48
49
50      Class methods:
51      --------------
52           atime, basename, blockdev?, catname, chardev?, chmod, chmod, chown,
53           compare, copy, ctime, delete, directory?, dirname, executable?,
54           executable_real?, exist?, exists?, expand_path, extname, file?,
55           fnmatch, fnmatch?, ftype, grpowned?, identical?, install, join,
56           lchmod, lchown, link, lstat, makedirs, move, mtime, new, owned?,
57           pipe?, readable?, readable_real?, readlink, rename, safe_unlink,
58           setgid?, setuid?, size, size?, socket?, split, stat, sticky?,
59           symlink, symlink?, syscopy, truncate, umask, unlink, utime,
60           writable?, writable_real?, zero?
61
62
63      Instance methods:
64      -----------------
65           atime, chmod, chown, ctime, flock, lstat, mtime, o_chmod, path,
66           truncate
67
68      linux3[2]%
```

To get more detailed information about a given method, simply specify the class and method as the argument:

```
linux3[2]% ri File.new

------------------------------------------------------------ File::new
     File.new(filename, mode="r")            => file
     File.new(filename [, mode [, perm]])    => file
------------------------------------------------------------------------
     Opens the file named by _filename_ according to _mode_ (default is
     ``r'') and returns a new +File+ object. See the description of
     class +IO+ for a description of _mode_. The file mode may
     optionally be specified as a +Fixnum+ by _or_-ing together the
     flags (O_RDONLY etc, again described under +IO+). Optional
     permission bits may be given in _perm_. These mode and permission
     bits are platform dependent; on Unix systems, see +open(2)+ for
     details.

        f = File.new("testfile", "r")
        f = File.new("newfile",  "w+")
        f = File.new("newfile", File::CREAT|File::TRUNC|File::RDWR, 0644)

linux3[3]%
```

Invoking the Interpreter

Not to be confused with this "shebang".

When running a ruby script (or Perl, Python, Shell, Tcl, etc...) on a UNIX based system (including Mac OS X) the first line of the script is very important in determining which program will execute it.

1. First, on UNIX based systems the file needs to be marked executable (`chmod u+x filename.rb`)
2. Next, when run from the shell, it reads the "shebang" line and the remainder of the file is fed to the appropriate interpreter
3. From there Ruby reads the entire script and begins execution

On a windows based-machine, typically files with an `.rb` extension are associated with the ruby interpreter and when launched will load with that program.

Alternatively, regardless of platform you can always type `ruby filename.rb` at your shell or command prompt. Since the `#` symbol is a comment in Ruby (and many other scripting languages), the line is simply ignored by the interpreter.

Ruby on GL

On GL, Ruby 1.9.1 is installed in `/usr/local/bin`...

```
1   linux1[1]% whereis ruby
2   ruby: /usr/local/bin/ruby /usr/local/lib/ruby
3   linux1[2]% /usr/local/bin/ruby --version
4   ruby 1.9.2p290 (2011-07-09 revision 32553) [i686-linux]
5   linux1[3]%
```

Sample Script

The following script is a basic example:

```
1    #!/usr/bin/env ruby
2    # The first line, starting with the "shebang" invokes the Ruby interpreter
3    # anything after a '#' is a comment
4
5    pattern = ARGV.shift                # shift first arg into a variable
6
7    while line = gets                   # while there is input, one line at a time
8        line.chomp                      # rip off the newline character
9        next if line =~ /^\#/           # skip lines that start with a comment
10       if /#{pattern}/                 # print if pattern was found
11           puts "found: " + line
12       end
13   end
```

What is Python

Modern programming language created by Guido van Rossum in 1990-1991

Python got its name from "Monty Python's Flying Circus"

Open source project managed by the Python Software Foundation

Cross platform scripting language

Python is an interpreted, object-oriented programming language

Python is dynamically type-checked and uses garbage collection for memory management

Python syntax is very different than other languages we have looked at — you will find yourself writing a lot of colons, and almost never writing a curly brace or semicolon

Python is a bit more strongly typed than that of JavaScript or PHP

Applications of Python

System Utilities — portable command line tools, testing systems

Internet Scripting — CGI programming, Java applets, XML, ASP, email tools

Django web framework gaining popularity

Graphical User Interfaces — Tk, MFC, Gtk, Qt, wx, etc...

Component Integration — C/C++ library front-ends

Database Access — persistent object stores, SQL database interfaces

Distributed Programming — client/server APIs like CORBA, COM

Rapid Prototyping/Development — throwaway or deliverable prototypes

Language Based Modules — replacing special purpose parsers with Python

More — Image processing, Numerical Programming, AI, etc...

Applications Written in Python

The following is a list of applications written (at least in part) with Python:

BitTorrent — tool for distributed download.

Plone — a popular open source CMS

Amarok — Linux media player

Blender — 3D graphics application, uses Python for embedded scripting

GIMP — uses Python for embedded scripting

Scribus — uses Python for embedded scripting

Freevo — Linux DVR software.

Fedora Config Tools — all the `system-config-*` tools for Fedora Core Linux

Anaconda — Red Hat/Fedora Installation Program

Sample Program

Below is a basic hello world example. Even with its simplicity, there are already some basic observations to be made:

Shebang line — dynamically gets the python install which is first in path
No sigil prefix — variable need not be prefixed with any characters (C style)
No semicolons — don't need them
print — automatically gives us a newline

```
1   #!/usr/bin/env python
2
3   # assign into string
4   str = "Hello World!"
5
6   # output string
7   print(str)
```

Output:

```
1   linux2[1]% hello.py
2   Hello World!
3   linux2[2]%
```

Interactive Interpreter

You can also run the Python interpreter in an interactive mode. Simply type `python` with no args and it will return to you a prompt where you can enter code. Press `Control-D` to exit.

```
1   linux2[1]% python
2   Python 2.4.3 (#1, Dec 11 2006, 11:38:52)
3   [GCC 4.1.1 20061130 (Red Hat 4.1.1-43)] on linux2
4   Type "help", "copyright", "credits" or "license" for more information.
5   >>> print "Hello interactive world!"
6   Hello interactive world!
7   >>> 5 * 5
8   25
9   >>>
10  linux2[2]%
```

Python Versions

There are currently 2 major, supported versions of Python out there. Python 2.7.x is the last version in the Python 2.x series — no new features will be added to 2.x (however security updates and bug fixes will continue to be published). Python 3.x breaks backwards compatibility with the Python 2.x series.

Looking around on GL, there appear to be several versions of Python available:

```
1   linux2[1]% whereis python
2   python: /usr/bin/python2.6-config /usr/bin/python2.6 /usr/bin/python
3   /usr/lib/python2.6 /usr/local/bin/python /usr/local/bin/python2.4
4   /usr/local/bin/python2.6 /usr/local/bin/python2.6-config
5   /usr/local/lib/python2.2 /usr/local/lib/python2.4
6   /usr/local/lib/python2.6 /usr/include/python2.6
7   /usr/share/man/man1/python.1.gz
8   linux2[2]%
```

What is JavaScript?

A scripting language that typically runs in a scripting engine inside of a web browser

JavaScript was originally designed to add interactivity to HTML pages

JavaScript is a cornerstone of modern web development

JavaScript is one of the more misunderstood programming languages

JavaScript is a fully-featured, dynamic, loosely-typed, prototype-based, language featuring first-class functions

JavaScript vs. Java

There is no relationship between Java and JavaScript. They are both programming languages that share a similar name (and some similar syntax), and that's about the extent of it. Java is a stand-alone, strongly-typed systems language whereas JavaScript is a loosely-typed, typically embedded in HTML and run in a browser scripting language.

“Java and JavaScript are similar like Car and Carpet are similar”

History

In the early days of the Web and HTML, Netscape realized that “rudimentary” scripting would improve the medium.

The initial JavaScript implementation was created in 1996 and released with Netscape 2.0

Designed to parse any part of the document and affect changes in elements

Parsing was made available through the Document Object Model (DOM)

The language almost immediately turned over to the European Computer Manufacturers Association (ECMA) for standardization.

ECMA produced the ECMAScript standard, which closely matched the existing JavaScript implementation

Features were added over the years

Today there exist many JavaScript engines with subtle differences in implementation

JavaScript Capabilities

What you **can do** with JavaScript:

Put dynamic text into an HTML page
Can react to events — can be set to execute when something happens (i.e. page loaded or when a user clicks on an HTML element)
Can read and write HTML elements — change the content of an HTML element (not just form elements)
Can be used as a first pass to validate data — although a user could easily turn it off in web browser
Can detect the visitor's browser — detect if it is Mozilla/Firefox, MSIE, WebKit, Opera or something else
Can be used to create/manipulate cookies — store information on the client through a very controlled mechanism
Can reach back and talk to the web server without reloading the entire page
Many new things on the cusp, with emerging specifications/recommendations:
Advanced 2D (and even 3D) graphics
Geo-location
Offline Web Apps
Web SQL Databases
DOM Storage

What you **cannot do** with JavaScript:

Read from file system directly
Write to file system directly
Interact with other processes
Hide the implementation — source code is downloaded to client browser where they can easily view the source code (though it can be obfuscated)

Motivation

A couple of years ago, we probably would have skipped over JavaScript, however it has been on of the hottest scripting languages over the past year or so.

We will be looking at JavaScript, as it is one of the cornerstones of Ajax (a technological

approach to developing rich web applications). Ajax an approach for building rich web applications used my many of today's modern web apps.

Basic Example

This is a very basic example of JavaScript working on a form, responding to user events, and popping up information.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <title>Hello</title>
  </head>
  <body>
    <p>
      <label for="name">Enter your name:</label>
      <input type="text" name="name" id="name" />
      <input type="button" id="button" value="Hello" />
    </p>
    <script>
      document.getElementById('button').onclick = function() {
        alert('Hello ' + document.getElementById('name').value);
      }
    </script>
  </body>
</html>
```

Performance Tasks

CMSC308 Programming Languages

MODULE 3
Object-Oriented Programming

Activity 3: Auto Shop (Java)

Create MyOwnAutoShop class which contains the main() method. Perform the following within the main()
method.

- Create an instance of Sedan class and initialize all the fields with appropriate values. Use super(...) method in
the constructor for initializing the fields of the superclass.

- Create two instances of the Ford class and initialize all the fields with appropriate values. Use super(...)
method in the constructor for initializing the fields of the super class.
www.oumstudents.tk

- Create an instance of Car class and initialize all the fields with appropriate values.

Display the sale prices of all instance

Instruction:

File name format should be:

CMSC308m3_lastname_firstname_year&section

You can submit it anytime before the end of the semester but take note, i can see the date of your submission and any of your activities in our group on my end. Please be proactive. Keep safe and good luck.

Rubric:
Hands-On Activity rubric

Understanding Directed Assess

Basic Adobe Photoshop Hands-on Activity

The activity is about proficiency with the functions and application in Programming Languages.

Score	COMPLIANCE *Project Guidelines and Compliance*	KNOWLEDGE *Ability to use required tools*	PRESENTATION *Demonstration of objective or competency in presentation*	SKILL *Uses a number of elements and principles*	
Excellent **90-100**	Project guidelines followed completely and all the required elements are present	Shows ability to use a variety of tools expertly and effectively -- at an excellent level.	The objective or competency was executed with thorough and precise judgments concluding in a carefully crafted product, presentation, or behavior.	Student used a skillful number of elements and principles to present their information.	
Proficient **80-89**	Project guidelines are mostly complete and all required elements are present.	Shows ability to use essential tools capably and effectively -- at a proficient level.	The objective or competency was executed with discernable and Correct judgments concluding in a proficiently crafted product, presentation, or behavior.	Student used an effective number of elements and principles to present their information.	
Adequate **70-79**	There is a missing important project requirement, or a guideline not followed	Shows ability to use basic tools competently and effectively -- at a satisfactory level.	The objective or competency was executed with apparent and acceptable judgments concluding in a satisfactorily crafted product, presentation, or behavior.	Student used a minimal number of elements and principles to present their information.	
Limited **60-69**	There are missing 2 or more important project requirements, or project guidelines not followed	Did not demonstrate ability to use basic tools competently and effectively at a satisfactory level.	The objective or competency was executed with superficial and vague Judgments concluding in an indistinctly crafted product, presentation, or behavior.	Student used an Inadequate number of elements and principles to present their information .	
Final Score					

This rubric came from ***http://barkerwchs.weebly.com/rubrics.html*** and was modified to suit the needs of the course.

Learning Resources

Website:

- *https://learn.saylor.org/course/view.php?id=79&sectionid=780*
- *https://learn.saylor.org/course/view.php?id=79&sectionid=781*
- *https://learn.saylor.org/course/view.php?id=79&sectionid=782*
- *https://learn.saylor.org/course/view.php?id=79&sectionid=783*
- *https://learn.saylor.org/course/view.php?id=79&sectionid=784*
- *https://learn.saylor.org/course/view.php?id=79&sectionid=785*
- *https://learn.saylor.org/course/view.php?id=79&sectionid=786*
- *https://www.computerscience.org/resources/computer-programming-languages/*