CS 300: Programming Languages

Course Syllabus, Fall 2013

Contents

1 Course description

This course begins with an overview of programming languages syntax and semantics, computational models (i.e., programming paradigms), and concepts that span across languages. Focus is on concepts and fundamental abstractions that characterize various computational models. In the proces, you will gain a beginning-level programming expertise in: Haskell, Lisp, Python, C, and C++.

This is a writing-intensive course and carries **writing-intensive designation**. Therefore, substantial writing is involved in this course. Writing assignments comprise 60% of the course grade. **Informal**, **low-** and **medium-stakes**, as well as **high-stakes** assignments are designed to effect learning through writing. Students are exposed to concepts, principles, techniques of programming languages through these writing assignments.

One or two informal, ungraded writing activity takes place in every class meeting. Low-stakes writing takes place at least once a week, and so does the medium-stakes writing. High-stakes writing is spread through the entire semester and involves document revision based on instructor and peer feedback.

2 Course Catalog description

Comparative study of the concepts found in contemporary programming languages. Emphasis is on design and evaluation of a language in terms of its features and their implementation. (PR: CS 210)

3 Instructor information and office hours

- Dr. V.N. Gudivada, Gullickson Hall Room 207, Phone: 304-696-5452. Please use Blackboard email for all communication related to this course.
- Course meets on TuTh 11:00 AM – 12:15 PM in GH 206A.
- Office hours:
  - Monday: 12:00 Noon - 2.00 PM
  - Tuesday: 10:00 AM - 11:00 AM and 1:00 PM - 2:.00 PM
  - Thursday: 10:00 AM - 11:00 AM and 1.00 PM - 2.00 PM
  - Other times by appointment

4 Prerequisites

- CS 210 (Algorithm Analysis and Design)

5 Course topics

- Computational models: imperative and declarative
- Programming languages syntax and semantics
- Names, scopes, and binding
- Control flow and control abstractions
- Data types and data abstractions
- Subprograms, modules, exceptions, and polymorphism
- Haskell, Lisp, Python, C, and C++.

6 BSCS degree program goals

a. an ability to apply knowledge of computing and mathematics appropriate to the discipline, including the ability to analyze and evaluate performance tradeoffs of algorithms, data structures, and hardware solutions;

b. an ability to analyze a problem, and identify and define the computing requirements appropriate to its solution;

c. an ability to design, implement, and evaluate a computer-based system, process, component, or program, including software systems of varying complexity, to meet desired needs;

d. an ability to function effectively on teams to accomplish a common goal;

e. an understanding of professional, ethical, legal, security, and social issues and responsibilities;

f. an ability to communicate effectively, both written and oral, with a range of audiences;

g. an ability to analyze the local and global impact of computing on individuals, organizations, and society;

h. a recognition of the need for and an ability to engage in continuing professional development;

i. an ability to use current techniques, skills, and tools necessary for computing practice, including the ability of expressing algorithms in at least two of the most important computer languages currently in use in academia and industry.

7 Course goals and relationship to BSCS program goals

After successful completion of this course, students should be able to:

① Enhance their **writing skills and strategies** by developing several documents and computer programs in the context of learning programming languages concepts and principles (contributes to degree program goal f).

② Explain the fundamental principles that underlie all programming languages (contributes to degree program goal i).

③ Demonstrate understanding of the basic structure of programming languages including syntax, semantics, data and control abstractions, pointers, subprograms, modules, concurrency, exceptions, and polymorphism (contributes to degree program goals c and i).

④ Explain the concepts and fundamental abstractions that underlie the following languages and demonstrate programing expertise in them: Haskell (pure functional), Lisp (semi-functional), Python (multiple paradigms), C (imperative, procedural), and C++ (imperative, object-oriented).

⑤ Understand, analyze, and document features of a programming language through self-study (contributes to degree program goals f and h).

8 Course timeline

❍ Week 1

- ◆ Computational models for programming (imperative and declarative)

❍ Week 2 – 3

- Names, scopes, and bindings

- Week 4
  - Control flow

- Week 5
  - Data types
  - Midterm exam

- Week 6
  - Control abstractions

- Week 7
  - Data abstraction and object orientation

- Weeks 8 - 9
  - Haskell

- Week 10
  - Lisp

- Week 11
  - Python

- Weeks 12 - 14
  - C/C++
  - Final exam (12 December 2013, 10.15 AM - 12.15 PM)

9 Instructional materials

Required textbook:

[1] Michael L. Scott. *Programming Language Pragmatics*. Third. Morgan Kaufmann, 2009.

Lecture materials for other languages will be drawn from various resources listed in the Bibliography (section 15). Lecture slides and handouts will be made available on muOnline. You should also consult Web resources listed in section 9.

Web resources

- TIOBE Programming Community Index. http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html
- Brian Heinold. *An Introduction to Programming Using Python*. Available for free download under Creative Commons Attribution-Noncommercial-Share Alike 3.0 Unported License at http://faculty.msmary.edu/heinold/Introduction_to_Programming_Using_Python_Heinold.pdf
- Bryan O'Sullivan, Don Stewart, and John Goerzen. *Real World Haskell*. Available for free under the Creative Commons Attribution-NonCommercial 3.0 Unported License in PDF and HTML formats at http://book.realworldhaskell.org/
- *Learn You a Haskell for Great Good!* is a great book for learning Haskell. You can read it online at http://learnyouahaskell.com/
- Allen B. Downey, *Think Python: How to Think Like a Computer Scientist*, OReilly Media, 2012. Available for free under the Creative Commons Attribution-NonCommercial 3.0 Unported License in PDF and HTML formats at http://www.greenteapress.com/thinkpython/
- Peter Seibel, *Practical Common Lisp* (free online textbook). http://www.gigamonkeys.com/book/
- M. Ben-Ari. *Understanding Programming Languages* (free online textbook). http://www.freetechbooks.com/understanding-programming-languages-t657.html
- J.R. Fisher Prolog :- Tutorial
- William E. Shotts, *The Linux Command Line*. Available for fee download in PDF format under a Creative Commons license at http://sourceforge.net/projects/linuxcommand/files/TLCL/09.12/TLCL-09.12.pdf/download
- Learn Prolog Now — an introductory course to programming in Prolog (free online book)
- Project Euler
- Logic in Action Open Course Project
- Real World Prolog usage

10 Writing assignments

Substantial technical writing is required in this course. In fact, we will be writing on a regular basis. Writing assignments comprise 60% of the course grade. We approach writing as an effective means to learn the subject matter, rather than as a practice to improve our writing abilities. However, it is expected that our ability to write well will also improve as a byproduct because of the frequency and the amount of writing involved.

Technical writing is precise, concise, and comprehensive. Typically sentences are short and written in active voice. Long sentences are hard to read and also prone to multiple interpretations. Brevity and accuracy are the hallmarks of technical communication. Variation in sentences is good only when it makes sense. Convoluted means to introduce sentence variation just for the sake of it is considered a bad practice.

We will do **two types of writing**: **informal** and **formal**.

In every class we will do one or more *explain to the village idiot* type informal, ungraded writing. Village idiot is a person known for ignorance and lack of sophistication. We will write about technical concepts in plain English in a way that even a village idiot can understand. We take turns in sharing our writing with the class. Each informal writing activity is designed to take no more than three to four minutes of class time.

We will do **three types of formal writing**. All written assignments need to be turned in as PDF documents.

10.1 Low-stakes writing assignments

Low-stakes writing is administered in the form developing programming solutions using various languages. These assignments will take anywhere from 10 minutes to an hour. A small portion of the class time will be allocated for this activity. Unfinished work must be completed outside the classroom. Low-stakes assignments are not graded, but solutions will be discussed and shared with students.

10.2 Medium-stakes writing assignments

There will be several graded, medium-stakes programming assignments. On average, there will be one medium-stakes assignment in a two-week period. These assignments are 3 to 5 pages in length. A medium-stakes assignment typically requires several hours (in the range of 5 to 10) for completion.

They are completed outside the class period. We will write programs using the following languages:

- Haskell
- Lisp
- Python
- C
- C++

10.3 High-stakes writing assignments

This course requires two formal, high-stakes writing assignments:

- The primary goal of the first assignment is to assess a student's ability to explain basic concepts of the imperative programming paradigm. The second goal is to assess a student's ability to categorize programming languages using various facets.
- The goals for the second assignment is to assess students' ability to understand, analyze, illustrate, and summarize a functional programming language.

Details will be provided in separate handouts.

11 Course assessment

The course assessment components include: writing assignments (30%), programming assignments (30%), one midterm exam (@20%), and final exam (20%). Course grade is awarded based on the following scheme:

Score	Letter Grade
>= 90	A
>= 80 & < 90	B
>= 70 & < 80	C
>= 60 & < 70	D
< 60	F

12 Classroom etiquette

- Students are expected to show up for class on time and participate in the class constructively.
- Attendance will be taken at the beginning of the class. However, attendance has no bearing on students' course grade. Students are not penalized for not attending classes. However, they are responsible for turning in assignments and projects on time and taking exams as scheduled.
- During the class and exams, students should turn off all types of electronic gadgets including mobile/smart phones, iPhones, iPods, blackberries, laptops. These devices must remain out of sight for the entire duration of the class. Students who violate this policy will be asked to leave the classroom.
- No internet browsing is allowed in the class.

13 muOnline

It is important that students visit muOnline regularly for up-to-date information about the course. muOnline hosts all the course materials including assignments, handouts, lecture notes, and reading materials.

14 Policy for students with disabilities

Marshall University is committed to equal opportunity in education for all students, including those with physical, learning and psychological disabilities. University policy states that it is the responsibility of students with disabilities to contact the Office of Disabled Student Services (DSS) in Prichard Hall 117, phone 304-696-2271, to provide documentation of their disability. Following this, the DSS Coordinator will send a letter to each of the student's instructors outlining the academic accommodation he/she will need to ensure equality in classroom experiences, outside assignment, testing and grading. The instructor and student will meet to discuss how the accommodation(s) requested will be provided. For more information, please visit http://www.marshall.edu/disabled or contact Disabled Student Services Office at Prichard Hall 117, phone 304-696-2271.

15 Bibliography

[1] Richard Reese. *Understanding and Using C Pointers*. http://it-ebooks.info/book/2265/. O'Reilly, 2013.

[2] TIOBE Software. *TIOBE Programming Community Index*. http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html. 2013.

[3] Carlos A. Varela. *Programming Distributed Computing Systems*. The MIT Press, 2013.

[4] Allen B. Downey. *Think Python: How to Think Like a Computer Scientist*. O'Reilly Media, Inc., available for free under the Creative Commons Attribution-NonCommercial 3.0 Unported License in PDF and HTML formats at http://www.greenteapress.com/thinkpython/, 2012.

[5] David Griffiths and Dawn Griffiths. *Head First C*. http://it-ebooks.info/book/704/. O'Reilly, 2012.

[6] Brian Heinold. *An Introduction to Programming Using Python*. Creative Commons Attribution-Noncommercial-Share Alike 3.0 Unported License, http://faculty.msmary.edu/heinold/Introduction_to_Programming_Using_Python_Heinold.pdf, 2012.

[7] Mark J. Johnson. *A Concise Introduction to Programming in Python*. Chapman & Hall/CRC, 2012.

[8] Ben Klemens. *21st Century C: Tips from the New School*. O'Reilly, 2012.

[9] Huw Collingbourne. *The Book of Ruby: A Hands-On Guide for the Adventurous*. No Starch Press, 2011.

[10] Conrad Barski. *Land of Lisp: Learn to Program in Lisp, One Game at a Time!* No Starch Press, 2010.

[11] Vernon L. Ceder. *The Quick Python Book*. Manning Publications Co., 2010.

[12] Mark Lutz. *Programming Python*. Fourth. O'Reilly Media, Inc., 2010.

[13] Bruce A. Tate. *Seven Languages in Seven Weeks: A Pragmatic Guide to Learning Programming Languages*. The Pragmatic Bookshelf, 2010.

[14] Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit*. O'Reilly Media, 2009.

[15] David A. Black. *The Well-Grounded Rubyist*. Manning Publications Co., 2009.

[16] Jeri R. Hanley and Elliot B. Koffman. *Problem Solving and Program Design in C*. Sixth. Addison Wesley, 2009.

[17] Mark Lutz. *Learning Python*. Fourth. O'Reilly, 2009.

[18] Michael L. Scott. *Programming Language Pragmatics*. Third. Morgan Kaufmann, 2009.

[19] David Flanagan and Yukihiro Matsumoto. *The Ruby Programming Language*. O'Reilly Media, Inc., 2008.

[20] Daniel P. Friedman and Mitchell Wand. *Essentials of Programming Languages*. Third. The MIT Press, 2008.

[21] Kent D. Lee. *Programming Languages: An Active Learning Approach*. Springer, 2008.

[22] Bryan O'Sullivan, Don Stewart, and John Goerzen. *Real World Haskell*. O'Reilly, 2008.

[23] M. Ben-Ari. *Understanding Programming Languages*. http://www.freetechbooks.com/understanding-programming-languages-t657.html, 2006.

[24] David Ascher, Alex Martelli, and Anna Ravenscroft. *Python Cookbook*. Second. O'Reilly, 2005.

[25] Peter Prinz and Tony Crawford. *C in a Nutshell*. O'Reilly Media, Inc., 2005.

[26] Peter Seibel. *Practical Common Lisp*. Apress, 2005.

[27] Steve McConnell. *Code Complete: A Practical Handbook of Software Construction*. Microsoft Press, 2004.

[28] Peter Van Roy and Seif Haridi. *Concepts, Techniques, And Models Of Computer Programming*. The MIT Press, 2004.

[29] Brian W. Kernighan and Rob Pike. *The Practice of Programming*. Addison Wesley, 1999.

[30] Harold Abelson and Gerald Jay Sussman. *Structure And Interpretation Of Computer Programs*. Second. The MIT Press, 1996.

[31] K.N. King. *C Programming: A Modern Approach*. W.W. Norton, 1996.

[32] Leon S. Sterling and Ehud Y. Shapiro. *The Art of Prolog: Advanced Programming Techniques*. Second. The MIT Press, 1994.

[33] Brian W. Kernighan and Dennis M. Ritchie. *C Programming Language*. Second. Prentice Hall, 1988.