Quick Links: FAQs | Meeting Times | Labs (labs.html) | Textbook | Grading Policy | Academic Integrity | Students with Disabilities

Course Overview

This is a course in fundamental computing principles for students with little to no computing background. It covers the following topics:

- Programming constructs involving sequencing, selection, iteration, and recursion.
- Data organization such as arrays and lists.
- Use of abstraction in computing for data representation, computer organization, computer networks, functional decomposition, and application programming interfaces.
- Use of computational principles in problem-solving like divide and conquer, randomness, and concurrency.
- Classification of computational problems based on complexity, non-computable functions, and using heuristics to find reasonable solutions to complex problems.
- Social, ethical and legal issues associated with the development of new computational artifacts will also be discussed.

Meeting Times

Lecture 1 MWF 2:30 - 3:20, WEH 7500, Margaret Reid-Miller.
Lecture 2 MWF 3:30 - 4:20, WEH 7500, Margaret Reid-Miller.
Labs (also known as recitations), held by course TAs in GHC 5208 and GHC 5210.

You are **required** to go to your assigned lecture and lab. Since part of your course grade depends on lab participation, you must go to your assigned section to get lab credit.

Textbook

There is **no required textbook** for this course. We will provide lecture slides and draft lecture notes as needed. We will additionally assign some readings from the book *Blown To Bits: Your Life, Liberty, and Happiness after the Digital Explosion* by Hal Abelson, Ken Ledeen, and Harry Lewis. Publisher: Addison-Wesley, 2008 (ISBN: 978-0137135592). This book is available **for free** from bitsbook.com (http://www.bitsbook.com/). You can buy a hard copy of this book if you wish from any major bookseller.

The following textbooks are recommended as supplementary resources:

- *Explorations in Computing: An Introduction to Computer Science and Python Programming* by John Conery. September 2014 edition. Publisher: Chapman and Hall/CRC. This book is available as an e-book or e-book rental (ISBN 9781498718349) and as a hardback (ISBN 9781466572447).

- *Introduction to Computation and Programming Using Python* by John V. Guttag. August 2013 edition. Publisher: MIT Press. This book is available as an e-book (ISBN 9780262316644) and as a paperback (ISBN 9780262525008).

Assignments

There are two types of assignments you will complete in this course: problems sets (PS) and programming assignments (PA). Problem sets are written assignments that help you test your understanding of conceptual parts in this course. Programming assignments help you test your programming skills or use of other online tools presented in class. Problem sets are handed in in class and programming assignments are handed in using the tool Autolab (https://milkshark.ics.cs.cmu.edu). You will be assigned about 11 problem sets and 9-10 programming assignments throughout the semester, *but* these numbers are subject to change.

Grading Policy

All assignments must be handed in on time (unless you are given instructions otherwise). **Late or missing work will receive zero credit.** The reason this is done is so that we can get feedback to you as quickly as possible so you can learn from your mistakes and prepare for the exams. Additionally, it allows us to post a sample solution as soon as possible for the benefit of all students. **We will drop 1 written assignment and 1 programming assignment with the lowest grade (except where noted).** You are required to provide documentation if you could not submit an assignment due to a legitimate reason (e.g. major illness, death in immediate family, university-sanctioned event with verification from advisor or coach, etc.). You must take all exams (written and lab exams) at the times they are given. There will be **NO MAKEUPS FOR EXAMS** allowed except for acceptable documented circumstances such as the ones listed above.
Your course grade will be calculated based on the following:

Homework Assignments: 30%
Lab Participation: 5%
Two Lab Exams: 10% (5% each)
Two Written Exams: 30% (15% each)
Final Exam: 25%

Grades from all assignments and exams may be reviewed for up to 5 days after they are returned and posted. After this period, the grade is considered final and cannot be changed. We reserve the right to review an entire assignment or exam if it is submitted for re-grading.

We use Autolab (https://milkshark.ics.cs.cmu.edu) for releasing all of your grades and we expect you to use it to keep track of your grade status.

Discussion Forum

We will use Piazza (https://piazza.com) for course announcements and online discussions.

Academic Integrity

The value of your degree depends on the academic integrity of yourself and your peers in each of your classes. Please read the University Policy on Academic Integrity (http://www.cmu.edu/policies/documents/Academic%20Integrity.htm) carefully to understand the penalties associated with academic dishonesty at Carnegie Mellon University.

Academic integrity means that any work you submit for this course is your own. This is critical to your learning. The policy's intention is that you never hand in something you don't understand. Your understanding must be deep enough that, if necessary, you could re-do the work completely on your own. In short, do your own work.

We want you to collaborate with other students *only if the collaboration improves your understanding*. Therefore, you can talk about the assignments, but no one may take notes or record the discussion. When you write your solution, it should be *yours*. Go to a separate area and write your own code or answers. Do

this *individually* so that you don't end up copying someone else's work. Your own solution, *even if it is incorrect*, is much better than someone else's that you don't understand.

When working on programming assignments, do not look at other students' code or show them your own. If you need that kind of help, get it from the course staff. You may discuss your code at a conceptual level; for example, "do we need a loop for this purpose or just an if statement?". You may collaborate on code at a whiteboard, but you may not take notes or photographs; the purpose of the collaboration is to develop your understanding so that you can then solve the problem yourself, on your own.

If the course staff sees similarities between your work and that of another student, we will attempt to understand what happened. Usually this involves asking you to explain your work and how you did it, and to re-create the work or solve a related problem during our meeting.

For exams, your work must be your own with no communication between you and others (except course staff), and you may use only authorized materials.

Often students have trouble keeping up with the workload due to personal issues. If this happens to you, your best action is to see your instructors. We can help you work toward a solution and will be happy to assist.

In this class, *cheating*, *copying*, or *plagiarism* means copying all or part of a program or homework solution from another student or unauthorized source, or knowingly giving such information to another student, or handing in a copy of work that you and another student did together, or giving or receiving unauthorized information during an examination. If you use information from another authoritative resource, you must cite the source of this information (and receive permission if required).

Students who violate this policy will be charged with academic dishonesty that can result in failure in this course and possible expulsion from Carnegie Mellon University. Review the official University Code for more information.

Every student is required to sign and return the Academic Integrity Form (other/academic_integrity_form.pdf) within the first week of classes.

Students with Disabilities

Individuals with documented disabilities may be eligible to receive services and accomodations from CMU's Equal Opportunity Services (EOS) office. For more information, please contact Larry Powell, Manager of Disability Services at (412) 268-2013 (voice/TTY).

Frequently Asked Questions

- *I have **questions about my assignment that is due soon**. How do I get help?*
  You can go to one of the office hours to receive help from the TAs or the instructors. You can also post your question to Piazza (https://piazza.com/class#fall2016/15110) to get an online response.
- *How can I **find my grades?***
  All your grades are available on Autolab (https://autolab.andrew.cmu.edu/courses/15110-f16/assessments) upon being graded. We make every effort to make your grades available within a week of their due date.
- *Some of **my grades are missing** on Autolab. What do I need to do?*
  First, contact your TA (labs.html) and ask why all or some grades are missing. Next, if the issue is not resolved within a day or two, send an e-mail to the instructors (copy your TA) detailing which grades are missing and your lab section.
- *Can I **switch my recitation** section?*

Yes, as long as we have enough space in the new recitation. You must make an official change of recitations through SIO.

- *I **must be out of town** for university related event (e.g. member of a team). What should I do about my assignments?*
  If you have an *official* excuse we will make special arrangements for you to submit the assignment; please contact the instructors.

- *I **missed one of the lab sessions**. Can I still submit my lab assignment late?*
  No. However, you can drop two labs and we will count this as one of them.

- *I am **out of town attending a family/important event** . How can I submit my assignments due for the week?*
  The programming assignment must be submitted online using Autolab if the submission link is active. Otherwise, you should alert the instructors and your TA that the link is not active and that you need to submit your work. The written assignment must be scanned and sent as an attachment to your TA before the due date. However, we can only allow you to do this one time during the semester.

- *I **missed the in class exam** because I fell sick. What should I do?*
  You must immediately seek medical treatment and receive an official medical excuse. You must also contact the instructors *prior* to the exam or as soon as possible. In this case, once we see the excuse, we can make arrangements to give you a makeup test. Otherwise, we will be unable to make any exceptions.

- *I need to **discuss my performance** in the course with someone. What should I do?*
  You must immediately contact one of the instructors of the course. They will be able to assist you in dealing with the situation.

- *What is the **best way to prepare** for an in class exam?*
  If you are attending lectures and doing homeworks, you are well prepared. All you need to do is to review all lectures and class assignments. We also regularly offer help sessions before the exam. Plan to attend one of them.

- *I am **failing the course**. Is there any extra work I can do to get a passing grade?*
  Unfortunately, in a large class like ours, we cannot make exceptions. The best way to avoid this situation is to talk to one of the instructors as soon as possible to find out what you need to do. Do not wait until the last few weeks of classes to discuss your performance.

- *I want to **add this course**. Is it possible to do it?*
  You can only add a course during the first two weeks of classes. We do not accept any new students after the second week. However, you are welcome to audit the course, provided we have enough space. Please consult an instructor.

120_1

CO120.1-Programming I

Course Aims

Academic aims

This is the students' first programming course. It aims to introduce some of the fundamentals of programming for beginners using a strongly-typed functional programming language (Haskell) that most students will have little or no experience of. The emphasis is on writing succinct, beautiful code, without being bogged down in the syntax and semantics of a conventional procedural or object-oriented language. The course is taught using a problem-solving approach with students being encouraged to use the various language features to solve well-specified problems independently and explore some fundamental algorithms and data structures in computer science.

Learning Outcomes

Students will be able to:

* Use static types as a partial specification of what a function is intended to do.

* Use type error messages as an aid to debugging.

* Devise recursive solutions to problems that involve iteration.

* Reason about the complexity of basic recursive functions and functions defined over linear and tree-like data structures.

* Design and use test suites for functional testing.

* Use the knowledge and experience gained to develop succinct and efficient solutions to unseen, but well-specified, problems of small to medium scale.

Course syllabus

Syllabus

Expressions, basic types, product types, arithmetic sequences, list comprehensions, function types and user-defined function definitions, recursion, polymorphism, list processing, enumerated types, higher-order functions, algebraic data types, type classes and overloading.

Pre-requisites

None, other than basic pre-university mathematics.

Teaching methods

Weekly lectures, catch-up tutorials, small-group tutorials, timetabled laboratory sessions, supervised catch-up laboratory sessions.

There is also a series of optional lectures on Advanced Programming in Haskell.

Assessments

There is an unassessed practice test (formative assessment only), a 'driving test' (20%) and a final 'main test' (80%), all of which are taken in the laboratory under exam conditions using the Lexis test administration system.

Students can also undertake independent self assessment through unassessed exercises, for which model answers are made available.

Reading list

S. Thompson, "Haskell: The Craft of Functional Programming" (Third Edition), Addison Wesley, 2011.

P. Hudak, "The Haskell School of Expression", Cambridge University Press, 2000.

R. Bird, "Introduction to Functional Programming using Haskell", Prentice Hall, 1988.

S. S. Skiena and M. A. Revilla, "Programming Challenges -- the Programming Contest Training Manual", Sptringer, 2003.

B. O'Sullivan, J. and D. B. Stewart, "Real World Haskell", O'Reilly Media, 2008.

The Haskell Wiki: http://haskell.org/

Course leaders

Dr Anthony Field

Imperial College London

South Kensington Campus
London SW7 2AZ, UK
tel: +44 (0)20 7589 5111

Campuses & maps >

Skip to Main Content

Course Description

CIS*1500 Introduction to Programming F,W (3-2) [0.50]

Introductory problem-solving, programming and data organization techniques required for applications using a general purpose programming language. Topics include control structures, data representation and manipulation, program logic, development and testing. For students who require a good understanding of programming or are planning on taking additional specialist Computing and Information Science courses. This is the entry point to all CIS programs.

Offering(s): Also offered through Distance Education format.

Restriction(s): CIS*1650

Department(s): School of Computer Science

Handbook

COMP10001 Foundations of Computing

On this page:

Print - /view/2016/COMP10001?output=PDF

Credit Points: 12.5

Level: 1 (Undergraduate)

Dates & Locations:

This subject has the following teaching availabilities in 2016:

Semester 1, Parkville - Taught on campus. Show/hide details - #

Pre-teaching Period Start	not applicable
Teaching Period	29-Feb-2016 to 29-May-2016
Assessment Period End	24-Jun-2016
Last date to Self-Enrol	11-Mar-2016
Census Date	31-Mar-2016
Last date to Withdraw without fail	06-May-2016

Semester 2, Parkville - Taught on campus. Show/hide details - #

Pre-teaching Period Start	not applicable
Teaching Period	25-Jul-2016 to 23-Oct-2016
Assessment Period End	18-Nov-2016
Last date to Self-Enrol	05-Aug-2016
Census Date	31-Aug-2016
Last date to Withdraw without fail	23-Sep-2016

Time Commitment:

Timetable can be viewed here - https://sws.unimelb.edu.au/2016/Reports/List.aspx?objects=COMP10001&weeks=1-52&days=1-7&periods=1-56&template=module_by_group_list.
For information about these dates, click here - http://ask.unimelb.edu.au/app/answers/detail/a_id/6032.
Contact Hours: 60 hours, comprised of three 1-hour lectures and one 2-hour workshop per week
Total Time Commitment:

170 hours

Prerequisites: None

Corequisites: None

Recommended Background Knowledge: None

Subject
INFO10001 Informatics 1: Data on the Web - /view/2016/INFO10001

Non Allowed Subjects:	INFO10001 Informatics-1:Practical Computing (prior to 2011) 615-145 Concepts of Software Development 1 433-151 Introduction to Programming (Advanced) 433-171 Introduction to Programming 600-151 Informatics-1: Practical Computing
Core Participation Requirements:	For the purposes of considering request for Reasonable Adjustments under the Disability Standards for Education (Cwth 2005), and Student Support and Engagement Policy, academic requirements for this subject are articulated in the Subject Overview, Learning Outcomes, Assessment and Generic Skills sections of this entry. It is University policy to take all reasonable steps to minimise the impact of disability upon academic study, and reasonable adjustments will be made to enhance a student's participation in the University's programs. Students who feel their disability may impact on meeting the requirements of this subject are encouraged to discuss this matter with a Faculty Student Adviser and Student Equity and Disability Support: http://services.unimelb.edu.au/disability - http://services.unimelb.edu.au/disability

Coordinator

Prof Christopher Leckie, Prof Tim Baldwin

Contact

Semester 1: Professor Tim Baldwin

email: tbaldwin@unimelb.edu.au - mailto:tbaldwin@unimelb.edu.au

Semester 2: A/Prof Chris Leckie

email: caleckie@unimelb.edu.au - mailto:bjpope@unimelb.edu.au

Subject Overview:	**AIMS** Solving problems in areas such as business, biology, physics, chemistry, engineering, humanities, and social sciences often requires manipulating, analysing, and visualising data through computer programming. This subject teaches students with little or no background in computer programming how to design and write basic programs using a high-level procedural programming language, and to solve simple problems using these skills. This subject is the first subject in the Computing & Software Systems and the Informatics majors, and introduces students to programming and the basics of algorithmic thinking. **INDICATIVE CONTENT** Fundamental programming constructs; fundamental data structures; abstraction; basic program structures; algorithmic problem solving, testing and debugging; introduction to the Web, multimedia and visualisation. Examples of projects that students complete are: • A text analytics “library” consisting of a series of independent functions to calculate/extract different things given a document/document collection as input • A video recommender system, broken down into a series of functions • An AI player for an online card game, designed such that students play off against each other (and against the class) at the end of semester.

Learning Outcomes:	**INTENDED LEARNING OUTCOMES (ILO)** On completion of this subject the student is expected to: 1. Use the fundamental programming constructs (sequence, alternation, selection) 2. Use the fundamental data structures (arrays, records, lists, associative arrays) 3. Use abstraction constructs such as functions 4. Understand and employ some basic program structures 5. Understand and employ some basic algorithmic problem solving techniques 6. Read, write, and debug simple, small programs
Assessment:	• A three-stage project, requiring approximately 30 - 35 hours of work, with stages due at the end of each third of the semester - approximately weeks 4, 8, and 12 (30%) • One 1-hour mid-semester test (10%) • A workshop assignment to demonstrate programming competency, due two thirds of the way through semester (10%), requiring approximately 10 - 13 hours of work per student • One 2-hour end-of-semester examination (50%). **Hurdle requirement:** To pass the subject, students must obtain at least: • 50% overall, 20/40 for the project and assignment work • And 30/60 for the mid-semester test and end-of-semester written examination combined. Intended Learning Outcomes (ILOs) 1-6 are addressed in the projects, the mid-semester test, and the workshop assignment and the final exam.
Prescribed Texts:	None
Breadth Options:	This subject potentially can be taken as a breadth subject component for the following courses: • Bachelor of Arts - https://handbook.unimelb.edu.au/view/2016/B-ARTS • Bachelor of Commerce - https://handbook.unimelb.edu.au/view/2016/B-COM • Bachelor of Environments - https://handbook.unimelb.edu.au/view/2016/B-ENVS • Bachelor of Music - https://handbook.unimelb.edu.au/view/2016/B-MUS You should visit learn more about breadth subjects - http://breadth.unimelb.edu.au/breadth/info/index.html and read the breadth requirements for your degree, and should discuss your choice with your student adviser, before deciding on your subjects.
Fees Information:	Subject EFTSL, Level, Discipline & Census Date - http://enrolment.unimelb.edu.au/fees
Generic Skills:	On completion of this subject, students should have developed the following generic skills: • An ability to apply knowledge of basic science and engineering fundamentals • An ability to undertake problem identification, formulation and solution • The capacity to solve problems, including the collection and evaluation of information • The capacity for critical and independent thought and reflection • An expectation of the need to undertake lifelong learning, and the capacity to do so.
	LEARNING AND TEACHING METHODS The subject is delivered through a combination of lectures and workshops (combination of tutorial and individual/group work in a computer lab). Students get

Notes:	a hands-on introduction to Python through a series of online worksheets with embedded programming tasks/automatic assessment, and then go on to complete three projects. **INDICATIVE KEY LEARNING RESOURCES** Students have access to lecture notes, lecture slides, tutorial worksheets, which houses the interactive worksheets as well as a programming environment. The subject LMS site also contains links to recommended resources relating to basic programming, and advanced problems for students who want to extend themselves. **CAREERS / INDUSTRY LINKS** As an introductory programming subject, this is relevant to all aspects of the IT industry. Exemplar companies/organisations which have been involved in the delivery of the subject (through guest lectures etc.) are: Palantir Technologies (software engineering, intelligent systems), AURIN (Australian Urban Research Infrastructure Network: geomatics, distributed computing, web development), VLSCI (Victorian Life Sciences Computing Initiative; computational biology, bioinformatics, distributed computing, big data). There have also been guest lecturers from within the university in fields including computational ophthalmology, electronic voting, and social media analysis.
Related Course(s):	Bachelor of Biomedicine - /view/2016/B-BMED Diploma in Informatics - /view/2016/D-INFO
Related Majors/Minors/Specialisations:	Science-credited subjects - new generation B-SCI and B-ENG. - /view/2016/%21B-SCI-SPC%2B1021 Selective subjects for B-BMED - /view/2016/%21B-BMED-SPC%2B1000

Date created: *11 October 2007* **Last modified:** *22 January 2009* **Authoriser:** *Vice-Principal and Academic Registrar, Office of the Vice-Principal and Academic Registrar* **Maintainer:** *Handbook Team, Applications Management, Academic Services Applications Development* **Help:** *for Future Students* *for Current Students*

Michaelmas Term 2016: Part IA lectures

Paper 1: Foundations of Computer Science

Lecturer: Professor L.C. Paulson

No. of lectures and practicals: 12 + 5 (NST and PBST students will take 4 practicals)

Suggested hours of supervisions: 3 to 4

This course is a prerequisite for Programming in Java and Prolog (Part IB).

Aims

The main aim of this course is to present the basic principles of programming. As the introductory course of the Computer Science Tripos, it caters for students from all backgrounds. To those who have had no programming experience, it will be comprehensible; to those experienced in languages such as C, it will attempt to correct any bad habits that they have learnt.

A further aim is to introduce the principles of data structures and algorithms. The course will emphasise the algorithmic side of programming, focusing on problem-solving rather than on hardware-level bits and bytes. Accordingly it will present basic algorithms for sorting, searching, etc., and discuss their efficiency using O-notation. Worked examples (such as polynomial arithmetic) will demonstrate how algorithmic ideas can be used to build efficient applications.

The course will use a functional language (ML). ML is particularly appropriate for inexperienced programmers, since a faulty program cannot crash. The course will present the elements of functional programming, such as curried and higher-order functions. But it will also introduce traditional (procedural) programming, such as assignments, arrays and references.

Lectures

- **Introduction to Programming.** The role of abstraction and representation. Introduction to integer and floating-point arithmetic. Declaring functions. Decisions and booleans. Example: integer exponentiation.
- **Recursion and Efficiency.** Examples: Exponentiation and summing integers. Overloading. Iteration *versus* recursion. Examples of growth rates. Dominance and O-Notation. The costs of some representative functions. Cost estimation.
- **Lists.** Basic list operations. Append. Naïve *versus* efficient functions for length and reverse. Strings.
- **More on lists.** The utilities `take` and `drop`. Pattern-matching: zip, unzip. A word on polymorphism. The “making change” example.
- **Sorting.** A random number generator. Insertion sort, mergesort, quicksort. Their efficiency.

- **Datatypes and trees.** Pattern-matching and case expressions. Exceptions. Binary tree traversal (conversion to lists): preorder, inorder, postorder.

- **Dictionaries and functional arrays.** Functional arrays. Dictionaries: association lists (slow) *versus* binary search trees. Problems with unbalanced trees.

- **Functions as values.** Nameless functions. Currying. The "apply to all" functional, `map`. *Examples*: matrix transpose and product. The predicate functionals `filter` and `exists`.

- **Sequences, or lazy lists.** Non-strict functions such as *IF*. Call-by-need *versus* call-by-name. Lazy lists. Their implementation in ML. Applications, for example Newton-Raphson square roots.

- **Queues and search strategies.** Depth-first search and its limitations. Breadth-first search (BFS). Implementing BFS using lists. An efficient representation of queues. Importance of efficient data representation.

- **Polynomial arithmetic.** Addition, multiplication of polynomials using ideas from sorting, etc.

- **Elements of procedural programming.** Address *versus* contents. Assignment *versus* binding. Own variables. Arrays, mutable or not. Introduction to linked lists.

Objectives

At the end of the course, students should

- be able to write simple ML programs;

- understand the fundamentals of using a data structure to represent some mathematical abstraction;

- be able to estimate the efficiency of simple algorithms, using the notions of average-case, worse-case and amortised costs;

- know the comparative advantages of insertion sort, quick sort and merge sort;

- understand binary search and binary search trees;

- know how to use currying and higher-order functions;

- understand how ML combines imperative and functional programming in a single language.

Recommended reading

* Paulson, L.C. (1996). *ML for the working programmer*. Cambridge University Press (2nd ed.).
Okasaki, C. (1998). *Purely functional data structures*. Cambridge University Press.

For reference only:
Gansner, E.R. & Reppy, J.H. (2004). *The Standard ML Basis Library*. Cambridge University Press. ISBN: 0521794781

Paper 1: Object-Oriented Programming

Lecturer: Dr R.K. Harle, Dr A.C. Rice and Dr S. Cummins

No. of lectures and practicals: 12 + 5

Suggested hours of supervisions: 3 to 4

Aims

The goal of this course is to provide students with the ability to write programs in Java and make use of the concepts of Object-Oriented Programming. Examples and discussions will use Java primarily, but other languages may be used to illustrate specific points where appropriate. The course is designed to accommodate students with diverse programming backgrounds; it is taught with a mixture of lectures and practical sessions where students can work at their own pace from a course handbook. Each practical class will culminate in an assessed exercise.

Lecture syllabus

- **Types, Objects and Classes** Moving from functional to imperative. Distinguishing state and behaviour. Primitive types. Function prototypes. Objects and classes as custom types. Introduction to parameterised types (templates/Generics).

- **Pointers, References and Memory** Pointers and references. The call stack and heap. Iteration and recursion. Pass-by-value and pass-by-reference. Objects as reference types in Java.

- **Creating Classes** Modularity. Encapsulation. Information hiding. Access modifiers. Advantages of immutability. Creating Generic types in Java. Static data.

- **Inheritance** Inheritance. Casting. Shadowing. Overloading. Overriding. Abstract Methods and Classes.

- **Polymorphism and Multiple Inheritance** Polymorphism in ML and Java. Multiple inheritance. Interfaces in Java.

COS 126: General Computer Science: An Interdisciplinary Approach

An introduction to computer science in the context of scientific, engineering, and commercial applications. The course will teach basic principles and practical issues, and will prepare students to use computers effectively for applications in computer science, physics, biology, chemistry, engineering, and other disciplines. Topics include: hardware and software systems; programming in Java; algorithms and data structures; fundamental principles of computation; and scientific computing, including simulation, optimization, and data analysis. No prior programming experience required. Video lectures, one or two classes, two preceptorials.

Semester: Fall16
Lectures: Tuesday,Thursday, 10:00-10:50
Location: TBD

Faculty

Kevin Wayne
Office: Computer Science 207
Extension: 4455
Email: wayne

Additional Information

Registrar's Fall16 COS offerings (http://registrar.princeton.edu/course-offerings/search_results.xml?subject=COS&term=1172)
CS Course Schedule (/courses/schedule)

The Undergraduate Coordinator is Colleen Kenny-McGinley
Email: ckenny
Office: Computer Science 210
Extension: 1746

Events (/general/newsevents/events)

15 Sep — Dan Boneh: New research directions in cryptography (/events/25422)

26 Sep — Frank Pfenning: Colloquium Speaker (/events/25423)

5 Oct — Tandy Warnow: Colloquium Speaker (/events/25424)

Full calendar and archive (/general/newsevents/events)

News (/general/newsevents/news)

AUGUST 16TH, 2016
Prof. Zhandry awarded Best Young Researcher at CRYPTO 2016 (/news/prof-zhandry-awarded-best-young-researcher-crypto-2016)

AUGUST 11TH, 2016
Strong showing of Princeton researchers at CCS (/news/strong-showing-princeton-researchers-ccs)

AUGUST 5TH, 2016
Prof. Martonosi named Cornell's Andrew D. White Professor-at-Large (/news/prof-martonosi-named-cornells-andrew-d-white-professor-large)

News archive (/general/newsevents/news)

Follow us: (https://www.facebook.com/PrincetonCS) (https://twitter.com/princetoncs)

(http://www.princeton.edu)

Terms of Use (/general/policies) | Privacy Policy (/general/policies#privacy) | Site Map (/sitemap)

(http://www.princeton.edu/engineering)

CSC108H1: Introduction to Computer Programming

Division
Faculty of Arts and Science

Course Description
Programming in a language such as Python. Elementary data types, lists, maps. Program structure: control flow, functions, classes, objects, methods. Algorithms and problem solving. Searching, sorting, and complexity. Unit testing. No prior programming experience required.

NOTE: You may not take this course concurrently with CSC120H1/CSC148H1, but you may take CSC148H1 after CSC108H1.

Department
Computer Science

Exclusion
CSC120H1, CSC121H1, CSC148H1

Course Level
100/A

Arts and Science Breadth
(5) The Physical and Mathematical Universes

Arts and Science Distribution
Science

Campus
St. George

Term
2016 Fall

Course Meeting Sections A&S Timetable Information

Activity	Day and Time	Instructor	Location	Class Size	Current Enrolment	Option to Waitlist	Delivery Mode
Lec 0101	FRIDAY 10:00-11:00 WEDNESDAY 10:00-11:00 MONDAY 10:00-11:00	E De Lara	WB 116 WB 116 WB 116	220	220	✔	IN-CLASS
Lec 0102	MONDAY 10:00-11:00 WEDNESDAY 10:00-11:00 FRIDAY 10:00-11:00	J Smith	HS 610 AH 100 AH 100	220	220	✔	IN-CLASS
Lec 0201	FRIDAY 13:00-14:00 WEDNESDAY 13:00-14:00 MONDAY 13:00-14:00	E De Lara	MB 128 MB 128 MB 128	220	220	✔	IN-CLASS
Lec 5101	WEDNESDAY 18:00-21:00	J Smith	MS 3154	250	250	✔	IN-CLASS
Lec 9901		P Gries		250	250	✔	ONLINE

Last updated
2016-08-17 09:15:16.0

www.utoronto.ca | Contacts | University Switchboard: (416) 978-2011
University of Toronto, 27 King's College Circle, Toronto, Ontario, Canada M5S 1A1

ACORN

Send Feedback

Course Information

ENGG1111 Computer Programming and Applications

2012-13				
Instructor(s):	**Wong Kenneth**	(Class A)	No. of credit(s):	6
	Chui C K	(Class B)		
	Mitcheson George	(Class C)		
Recommended Learning Hours:	Lecture: 27 Lab Session: 12			
Pre-requisite(s):				
Co-requisite(s):				
Remarks:	This course may not be taken with CSIS1117 or ENGG1013 or ENGG1014.			

Course Learning Outcomes

1. Analyze simple problems and derive solutions, providing a logical flow of instructions
2. Use and construct functions for structured computer programs
3. Demonstrate competency in the use of various data structures in program writing
4. Identify and correct different types of programming errors, including data validation
5. Demonstrate competency in program testing and debugging

Syllabus

Calendar Entry:
This course covers both the basic and advanced features of the C/C++ programming languages, including syntax, identifiers, data types, control statements, functions, arrays, file access, objects and classes, class string, structures and pointers. It introduces programming techniques such as recursion, linked lists and dynamic data structures. The concept and skills of program design, implementation and debugging, with emphasis on problem-solving, will also be covered.

Target students are those who wish to complete the programming course in a more intensive mode in 1 semester. Students with some programming knowledge are encouraged to take this course.

Detailed Description:

Weeks 1-6 Basic programming.	**Mapped to Outcomes**
C++ syntax, identifiers, data types, control statements, functions, arrays, file access	1, 2
Program design, implementation and debugging, problem solving	1, 4, 5
Weeks 7-13: Advance features.	**Mapped to Outcomes**
Class and objects, string, structures and pointers, recursion, linked list, dynamic data structures	1, 2, 3

Assessment:
Continuous Assessment: 50%
Written Examination: 50%

Teaching Plan
Class ENGG1111A Class ENGG1111B Class ENGG1111C

PAGE TOP

Email: enquiry@cs.hku.hk Telephone: 2859 2180 Fax: 2559 8447

CS 137 Programming Principles

Objectives

This course introduces software engineering students to elementary data structures, and to the functional programming paradigm.

Intended Audience

Level 1A Software Engineering undergraduates. It is assumed that students have experience developing well-structured, modular programs.

Related Courses

Antirequisites: CS 115, 135, 136, 145, CHE 121, CIVE 121, ECE 150, GENE 121, PHYS 239, SYDE 121

Successor: CS 138.

Hardware and Software

Used in course: An integrated development environment (IDE) for C++ (such as xCode for Mac OS X).

References

C Programming: A Modern Approach 2nd ed. Author: K.N. King

Schedule

Three hours of lecture per week, plus a two-hour lab and a one-hour tutorial. Normally available in Fall only.

Outline

Introduction and review (6 hours)

Review of hardware and software, problem solving and programming, including declarations, selection and looping constructs, I/O.

Subprograms (6 hours)

Functions. Scoping. Parameter passing, by value and by reference. Top-down design. Programming with stubs.

Structured Data (7.5 hours)

Arrays. Structures. Arrays of structures. Multidimensional arrays. Appropriate choice of data structures. String processing and tokenization.

Recursion (3 hours)

Introduction to recursive procedures and functions.

Analysis (1.5 hours)

Introduction to O-notation, space and time efficiency, and complexity.

Sorting Algorithms (4.5 hours)

Algorithm design and efficiency, through discussion of of elementary sorting algorithms (insertion sort, selection sort) and advanced sorting algorithms (mergesort, quicksort, heapsort).

Pointers and Dynamic Structures (6 hours)

Introduction to pointers, memory allocation and deallocation, singly and doubly-linked lists.

History of Computer Science (1.5 hours)

Babbage, Hilbert, Godel, Church, Turing. Early development of electronic computers and programming languages. History of concepts covered in this course.

David R. Cheriton School of Computer Science
University of Waterloo
Waterloo, Ontario, Canada N2L 3G1

Tel: 519-888-4567 x33293
Fax: 519-885-1208

MICHIGAN ENGINEERING | COLLEGE ADMINISTRATION, 1221 BEAL AVENUE, ANN ARBOR, MI 48109-2102
+1 (734) 647-7000

Electrical Engineering and Computer Science Courses

100 Level Courses

EECS 101. Thriving in a Digital World
Prerequisite: none. (4 credits)
From mobile apps to bitmaps, this course explores computational technologies and how they impact society and our everyday lives. Topics include: social networks, creative computing, algorithms, security and digital privacy. Traditional computer programming is not a primary focus. Instead, mobile applications will be created using a novel visual programming environment.

EECS 183. Elementary Programming Concepts
Prerequisite: none. (Credit for only one: EECS 183, ENGR 101) (4 credits)
Fundamental concepts and skills of programming in a high-level

language. Flow of control: selection, iteration, subprograms. Data structures: strings, arrays, records, lists, tables. Algorithms using selection and iteration (decision making, finding maxima/minima, searching, sorting, simulation, etc.) Good program design, structure and style are emphasized. Testing and debugging. Not intended for Engineering students (who should take ENGR 101), nor for CS majors in LSA who qualify to enter EECS 280.

200 Level Courses

EECS 203 (CS 203). Discrete Mathematics
Prerequisite: MATH 115. (4 credits)
Introduction to the mathematical foundations of computer science. Topics covered include: propositional and predicate logic, set theory, function and relations, growth of functions and asymptotic notation, introduction to algorithms, elementary combinatorics and graph theory and discrete probability theory.

EECS 215. Introduction to Electronic Circuits
Prerequisite: MATH 116, ENGR 101, Corequisite PHYSICS 240 (or 260). Cannot receive credit for both EECS 314 and EECS 215. (4 credits)
Introduction to electronic circuits. Basic Concepts of voltage and current; Kirchhoff's voltage and current laws; Ohm's law; voltage and current sources; Thevenin and Norton equivalent circuits; DC and low frequency active circuits using operational amplifiers, diodes, and transistors; small signal analysis; energy and power. Time- and frequency-domain analysis of RLC circuits. Basic passive and active electronic filters. Laboratory experience with electrical signals and circuits.

EECS 216. Introduction to Signals and Systems
Prerequisite: EECS 215 or EECS 314 or BIOMEDE 211, preceded or accompanied by MATH 216. (4 credits).
Theory and practice of signals and systems engineering in continuous and discrete time. Continuous-time linear time-invariant systems, impulse response, convolution. Fourier series, Fourier transforms, spectrum, frequency response and filtering. Sampling leading to basic digital signal processing using the discrete-time Fourier and the discrete Fourier transform. Laplace transforms, transfer functions, poles and zeros, stability. Applications of Laplace transform theory to RLC circuit analysis. Introduction to communications, control and signal processing. Weekly recitations and hardware/Matlab software laboratories.

EECS 230. Electromagnetics I
Prerequisite: MATH 215, PHYS 240 (or 260), EECS 215. (4 credits)
Vector calculus. Electrostatics. Magnetostatics. Time-varying fields: Faraday's Law and displacement current. Maxwell's equations in differential form. Traveling waves and phasors. Uniform plane waves. Reflection and transmission at normal incidence. Transmission lines. Laboratory segment may include experiments with transmission lines, the use of computer-simulation exercises, and classroom demonstrations.

FECHAR X

Ementa de Disciplina

INF1004
PROGRAMACAO P/ INFORMATICA I
4 créditos

Ementa
Introdução à computação; programando com funções; condicionais e operadores lógicos; solução conceitual; introdução a iteração; modelo de computador; tipagem de dados, variáveis e operadores em uma linguagem procedimental; entrada e saída; controle de fluxo procedimental; funções; iteração; vetores e matrizes; desenvolvimento de programas. Vinculação de laboratório: INF 1006.

Bibliografia

CELES FILHO, WALDEMAR; CERQUEIRA, RENATO FONTOURA DE GUSMÃO; RANGEL NETTO, JOSÉ LUCAS MOURÃO. **INTRODUÇÃO À ESTRUTURAS DE DADOS: COM TÉCNICAS DE PROGRAMAÇÃO EM C**; RIO DE JANEIRO: CAMPUS, 2004.

KERNIGHAN, BRIAN W.; RITCHIE, DENNIS M. **C, A LINGUAGEM DE PROGRAMAÇÃO PADRÃO ANSI**; RIO DE JANEIRO: CAMPUS, 1989.

SCHILDT, HERBERT. **C COMPLETO E TOTAL**; SÃO PAULO: MAKRON, 1991.

Bibliografia Complementar
Nenhuma biliografia complementar encontrada para INF1004

Pré-requisitos
Nenhum pre-requisito encontrado para INF1004

Ultima atualização: 29/01/2014

Course Catalogue

252-0835-00L Computer Science I

Semester	Autumn Semester 2014
Lecturers	**F. O. Friedrich**
Periodicity	yearly course
Language of instruction	German

Tabs

Catalogue data

Abstract	The course covers the fundamental concepts of computer programming with a focus on systematic algorithmic problem solving. Teached language is C++. No programming experience is required.
Objective	Primary educational objective is to learn programming with C++. When successfully attended the course, students have a good command of the mechanisms to construct a program. They know the fundamental control and data structures and understand how an algorithmic problem is mapped to a computer program. They have an idea of what happens "behind the secenes" when a program is translated and executed. Secondary goals are an algorithmic computational thinking, undestanding the possibilities and limits of programming and to impart the way of thinking of a computer scientist.
Content	The course covers fundamental data types, expressions and statements, (Limits of) computer arithmetic, control statements, functions, arrays, structural types and pointers. The part on object orientiation deals with classes, inheritance and polymorphy, simple dynamic data types are introduced as examples. In general, the concepts provided in the course are motivated and illustrated with algorithms and applications.
Lecture notes	A script written in English will be provided during the semeter. The script and slides will be made available for download on the course web page.
Literature	Bjarne Stroustrup: Einführung in die Programmierung mit C++, Pearson Studium, 2010 Stephen Prata, C++ Primer Plus, Sixth Edition, Addison Wesley, 2012 Andrew Koenig and Barbara E. Moo: Accelerated C++, Addison-Wesley, 2000.
Prerequisites / Notice	From AS 2013, an admission to the exam does not any more formally require an attending of the recitation sessions. Handing in solutions to the weekly exercise sheets is thus not mandatory, but we strongly recommend it. Examination is a one hour-long written test.

Performance assessment

Performance assessment information (valid until the course unit is held again)

▸ **Performance assessment as a two-semester course together with 252-0836-00L Computer Science II (next semester)**

For programme regulations (Examination block)	Bachelor Programme in Computational Science and Engineering 2008; Version 01.08.2014 (Examination Block) Bachelor's Programme in Electrical Engineering and Information Technology 2004; Version 19.06.2012 (Examination Block) Bachelor's Programme in Electrical Engineering and Information Technology 2012; Version 24.02.2016 (Examination Block)
ECTS credits	8 credits

▸ **Performance assessment as a semester course (other programmes)**

In examination block for	Bachelor's Programme in Computational Science and Engineering 2010; Version 01.08.2014 (Examination Block) Bachelor's Programme in Computational Science and Engineering 2012; Version 24.02.2016 (Examination Block)
ECTS credits	4 credits
Examiners	**F. O. Friedrich**
Type	session examination
Language of examination	German
Course attendance confirmation required	No
Repetition	The performance assessment is offered every session. Repetition possible without re-enrolling for the course unit.
Mode of examination	written 60 minutes
Additional information on mode of examination	Wir offerieren im Semester zwei freiwillige Programmierübungen, welche korrigiert und bewertet werden. Die dabei erzielten Punkte werden in die spätere Prüfungsklausur als Bonus mitgenommen. Maximal erreichbarer Bonus entspricht 1/4 Note. Dieser Bonus kann nicht in spätere Repetitionsklausuren mitgenommen werden.
Written aids	keine

If the course unit is part of an examination block, the credits are allocated for the successful completion of the whole block.
This information can be updated until the beginning of the semester; information on the examination timetable is binding.

Learning materials

Main link	**Information**

Only public learning materials are listed.

Courses

Number	Title	Hours				Lecturers
252-0835-00 V	Informatik I	2 hrs	Wed	08-10	ETF E 1 »	**F. O. Friedrich**
252-0835-00 U	Informatik I	2 hrs	Mon	13-15	CHN F 42 »	**F. O. Friedrich**
				13-15	ETZ E 6 »	
				13-15	ETZ F 91 »	
				13-15	ETZ G 91 »	
				13-15	ETZ H 91 »	
				13-15	IFW A 34 »	
				13-15	ML F 34 »	
				13-16	HG E 26.1 »	
				15-17	CHN G 46 »	
				15-17	ETZ F 91 »	
				15-17	ETZ G 91 »	
				15-17	ETZ H 91 »	
				15-17	ETZ K 91 »	
				15-17	HG D 5.1 »	
				16-19	HG E 26.1 »	
				17-19	ETZ F 91 »	
				17-19	ETZ G 91 »	
				17-19	ETZ H 91 »	
				17-19	ETZ K 91 »	
				17-19	IFW D 42 »	
			27.10.	13-15	HG E 23 »	

Restrictions

There are no additional restrictions for the registration.

Offered in

Programme	Section	Type
Electrical Engineering and Information Technology Bachelor	**First Year Examinations**	**O**
Computer Science (General Courses)	**Computer Science for Non-Computer Scientists**	**Z**

Computational Science and Engineering Bachelor	First Year Courses	O

Imprint 17 August 2016 Version 2015.1 prod (prod red9)

Course 6: Electrical Engineering and Computer Science
Fall 2016

Course 6 Home *CI-M Subjects for Undergraduate Majors* *Evaluations (Certificates Required)*

| 6.00-6.299 | 6.30-6.799 | 6.80-6.ZZZ |

Basic Undergraduate Subjects

6.00 Introduction to Computer Science and Programming

(,) R
Prereq: None
Units: 3-7-2
Lecture: *MW3* (26-100) **Lab:** *TBA* ***+final***

Introduction to computer science and programming for students with little or no programming experience. Students learn how to program and how to use computational techniques to solve problems. Topics include software design, algorithms, data analysis, and simulation techniques. Assignments are done using the Python programming language. Meets with 6.0001 first half of term and 6.0002 second half of term. Credit cannot also be received for 6.0001 or 6.0002. Final given during final exam week.
J. V. Guttag
No textbook information available

6.0001 Introduction to Computer Science Programming in Python

(,); first half of term
Prereq: None
Units: 2-3-1
Ends Oct 21. **Lecture:** *MW3* (26-100) **Recitation:** *F10* (36-112, 36-153) or *F11* (36-112, 36-153) or *F12* (36-153) or *F1* (36-153) or *F2* (36-153) or *F3* (36-153) or *F12* (36-112) or *F1* (36-112) or *F2* (36-112)

Introduction to computer science and programming for students with little or no programming experience. Students develop skills to program and use computational techniques to solve problems. Topics include the notion of computation, Python, simple algorithms and data structures, testing and debugging, and algorithmic complexity. Combination of 6.0001 and 6.0002 counts as REST subject. Final given in the seventh week of the term.
J. V. Guttag
Textbooks (Fall 2016)

6.0002 Introduction to Computational Thinking and Data Science

(,); second half of term
Prereq: 6.0001 or permission of instructor
Units: 2-3-1
Begins Oct 24. **Lecture:** *MW3* (26-100) **Recitation:** *F10* (36-112, 36-153) or *F11* (36-112, 36-153) or *F12* (36-153) or *F1* (36-153) or *F2* (36-153) or *F3* (36-153) or *F12* (36-112) or *F1* (36-112) or *F2* (36-112) ***+final***

Provides an introduction to using computation to understand real-world phenomena. Topics include plotting, stochastic programs, probability and statistics, random walks, Monte Carlo simulations, modeling data, optimization problems, and clustering. Combination of 6.0001 and 6.0002 counts as REST subject. Final given during final exam week.
J. V. Guttag
Textbooks (Fall 2016)

6.002 Circuits and Electronics

(,) R
Prereq: Physics II (GIR); *Coreq: 18.03 or 2.087*
Units: 4-1-7
Lecture: *TR11* (34-101) **Lab:** *TBA* **Recitation:** *WF11* (26-310) or *WF12* (26-310) or *WF1* (26-310) or *WF2* (26-310) ***+final***

Fundamentals of the lumped circuit abstraction. Resistive elements and networks, independent and dependent sources,

Introduction to Computer Science Programming in Python or return to Course Catalog Search

6.0001 MIT

School
Massachusetts Institute of Technology

Department
Electrical Eng & Computer Sci

Faculty
Guttag, John V.

Term
Spring 2016 (show academic calendar)

Day and Time
Contact host school for schedule

Credits
6.00 (show credit conversion for other schools)

Credit Level
Undergraduate

Description
Introduction to computer science and programming for students with little or no programming experience. Students develop skills to program and use computational techniques to solve problems. Topics include the notion of computation, Python, simple algorithms and data structures, testing and debugging, and algorithmic complexity. Combination of 6.0001 and 6.0002 counts as REST subject.

Prerequisite(s)
None

Textbook Information Currently no textbook information is available for this course. Please check back or visit the Harvard Coop.

Cross Registration
Crossregistration for MIT courses requires instructor approval.

Please login to create your cross registration course list.

If you are a non-Harvard student, please log in with your **XID**.

Courses from the Massachusetts Institute of Technology are included to facilitate cross-registration by eligible Harvard students into MIT. These courses are taught at MIT, by MIT instructors, and are subject to MIT policies.

Hiroshima University Syllabus

Back to syllabus main page

Japanese

Academic Year	2016Year	**School/Graduate School**	Liberal Arts Education Program		
Lecture Code	63130001	**Subject Classification**	Foundation Courses		
Subject Name	プログラミング序説[1工二]				
Subject Name（Katakana）	プログラミングジヨセツ				
Subject Name in English	Introduction to Computer Programming				
Instructor	MORIKAWA KATSUMI,TAKAFUJI DAISUKE,KAMEI SAYAKA				
Instructor (Katakana)	モリカワ　カツミ,タカフジ　ダイスケ,カメイ　サヤカ				
Campus	HigashiHiroshima	**Semester/Term**	1st-Year, First Semester, First Semester		
Days, Periods, and Classrooms	(1st) Mon9-10：工103，メセ本端				
Lesson Style	Lecture	**Lesson Style【More Details】**	Lecture & practice (fifty-fifty)		
Credits	2	**Class Hours/Week**	2	**Language on Instruction**	J : Japanese
Course Level	1 : Undergraduate Introductory				
Course Area（Area）	02 : Informatics				
Course Area（Discipline）	02 : Computing Systems and Information Infrastructures				
Eligible Students	1st year students				
Keywords	Computer programming, C programming language.				
Special Subject for Teacher Education		**Special Subject**			
Related Programs					
Class Status within General Education /Integrated Courses					
Expected Outcome					
Class Objectives	Acquire basic programming techniques using C language, i.e., (1) learn how to write computer programs, (2) understand the behavior of				

/Class Outline	programs written in C, and (3) write and run C programs for simple processing requests.
Class Schedule	Lesson 1: Guidance. Introduction to programming. Linux and C programming Language. Steps of writing and running a program. (Morikawa) Lesson 2: Programming practice. (Takafuji) Lesson 3: Display "hello, world". Declare variables and assign values. Comments in code. (Morikawa) Lesson 4: Programming practice. (Takafuji) Lesson 5: Data types for integer, floating-point, and character. Formatted output using printf(). Error messages and their solutions. (Morikawa) Lesson 6: Programming practice. (Takafuji) Lesson 7: Test statement. if statement, if-else statement, Increment operator. for statement. (Morikawa) Lesson 8: Programming practice. (Takafuji) Lesson 9: while statement. do statement. Interchangeability among for, while, and do statements. Nested loops. Reading keyboard input. (Morikawa) Lesson 10: Programming practice. (Kamei) Lesson 11: Operations involving integer and floating-point types. Type conversions. Array. (Morikawa) Lesson 12: Programming practice. (Kamei) Lesson 13: Array of arrays. Array initialization. Practical problems using arrays. (Morikawa) Lesson 14: Programming practice. (Kamei) Lesson 15 Summary & final test. (Morikawa)
Text/Reference Books,etc.	(Textbook) Cの絵本，（株）アンク著，翔泳社 (Reference) 新・C言語入門 シニア編，林 晴比古 (著)，ソフトバンククリエイティブ
PC or AV used in Class,etc.	Textbook, handout, PowerPoint
Suggestions on Preparation and Review	For lessons 1 to 14: Topics studied in the classroom will be covered by the programming practice at ICE room scheduled next week. Try to input sample programs, compile & run these programs. Reading and understanding many good programs is a best way of mastering programming skills.
Requirements	(1) The rooms of this class are (i) lecture room for lessons, and (ii) ICE room for practice. Go to the appropriate room based on the schedule. (2) The contents of practice are how to write C programs, how to use tools for programming, and practices using several examples. (3) Each student should write your programs, obtain the results of these programs, write reports and submit them by the end of the given deadline.
Grading Method	The maximum score of all practices at ICE is about 35. The maximum score of final test is about 65. A pass grade is (i) 60 or more of the sum of them, and (ii) 50% or more of the final test.
Message	Each student should join the class positively to overcome difficulties and solve problems when mastering programming skills. Inactive students may fail to accomplish the class objectives.
	1. Assemble in lecture room 103 (Faculty of Engineering) on the first

Other	day of the class, i.e., 11 April (Mon). 2. This class accepts students of the Cluster II, Faculty of Engineering.
Please fill in the term-end class evaluation questionnaire. Instructors will reflect on your feedback and utilize the information for improving their teaching. Small classes will not conduct the questionnaire to prevent your identity from being uncovered.	

Back to syllabus main page

Computer Science and Engineering

COMP 1001 Exploring Multimedia and Internet Computing [3 Credit(s)]
This course is an introduction to computers and computing tools. It introduces the organization and basic working mechanism of a computer system, including the development of the trend of modern computer system. It covers the fundamentals of computer hardware design and software application development. The course emphasizes the application of the state-of-the-art software tools to solve problems and present solutions via a range of skills related to multimedia and internet computing tools such as internet, e-mail, WWW, webpage design, computer animation, spread sheet charts/figures, presentations with graphics and animations, etc. The course also covers business, accessibility, and relevant security issues in the use of computers and Internet. *Exclusion(s):* ISOM 2010, any COMP courses of 2000-level or above

COMP 1002 Computer and Programming Fundamentals I [3 Credit(s)]
Introduction to computers and programming. Computer hardware and software. Problem solving. Program design. Procedural abstraction. Debugging and testing. Simple and structured data types. Recursive programming. Introduction to searching and sorting. *Exclusion(s):* COMP 1004 (prior to 2013-14), COMP 1021, COMP 1022P, COMP 1022Q

COMP 1021 Introduction to Computer Science [3 Credit(s)]
This course introduces students to the world of Computer Science. Students will experience a range of fun and interesting areas from the world of computing, such as game programming, web programming, user interface design and computer graphics. These will be explored largely by programming in the Python language. *Exclusion(s):* COMP 1002, COMP 1004 (prior to 2013-14), COMP 1022P, COMP 1022Q, COMP 2011

COMP 1022P Introduction to Computing with Java [3 Credit(s)]
This course is designed to equip students with the fundamental concepts of programming elements and data abstraction using Java. Students will learn how to write procedural programs using variables, arrays, control statements, loops, recursion, data abstraction and objects using an integrated development environment. *Exclusion(s):* COMP 1002, COMP 1004 (prior to 2013-14), COMP 1021, COMP 1022Q, COMP 2011, ISOM 3320

COMP 1022Q Introduction to Computing with Excel VBA [3 Credit(s)]
This course is designed to equip students with the fundamental concepts of programming using the VBA programming language, within the context of the Microsoft Excel program. Students will first learn how to use Excel to analyze and present data, and will then learn how to use VBA code to build powerful programs. *Exclusion(s):* COMP 1002, COMP 1004 (prior to 2013-14), COMP 1021, COMP 1022P, COMP 2011, ISOM 3230

COMP 1029A Introduction to Mobile Application Development Using Android [1 Credit(s)]
[Previous Course Code(s): COMP 4901C*]* This course provides a basic introduction to mobile application development using the Android platform. It is intended for students who have some prior programming experience, but wish to learn the basics of mobile application development. The course will introduce them to the Android SDK and development environment, Android application components: Activities and their lifecycle, UI design, Multimedia, and 2D graphics support in Android. Students explore these concepts through self-learning course materials and guided laboratory exercises. Graded P or F. *Prerequisite(s):* COMP 1002 OR COMP 1004 (prior to 2013-14) OR COMP 1021 OR COMP 1022P OR COMP1022Q

COMP 1029C C Programming Bridging Course [1 Credit(s)]
This course introduces the C programming language. It is intended for students who already have some experience in computer programming but wish to learn how to apply those programming skills to the C language. The course covers basic programming topics, such as variables, control, loops, and functions, to more advanced topics. Students explore these by self-learning of course materials together with guided programming exercises. Students without the prerequisites but possess relevant programming knowledge may seek instructor's approval for enrolling in the course. Graded P or F. *Exclusion(s):* COMP 1002, COMP 1004 (prior to 2013-14), COMP 2011 *Prerequisite(s):* COMP 1021 OR COMP 1022P OR COMP 1022Q OR ISOM 3230 OR ISOM 3320

COMP 1029J Java Programming Bridging Course [1 Credit(s)]
This course introduces the Java programming language. It is intended for students who already have some experience in computer programming but wish to learn how to apply those programming skills to the Java language. The course covers basic programming topics such as variables, control statements, loops, functions, and object-oriented programming concepts. Students explore these by self-learning of course materials together with guided programming exercises. Students without the prerequisites but possess relevant programming knowledge may seek instructor's approval for enrolling in the course. Graded P or F. *Exclusion(s):* COMP 1022P, COMP 3021, ISOM 3320 *Prerequisite(s):* COMP 1002 OR COMP 1004 (prior to 2013-14) OR COMP 1021 OR COMP 1022Q OR ISOM 3230

Imperative Programming I: 2016-2017

Lecturer	**Geraint Jones (/people/Geraint.Jones)**
Degrees	**Preliminary Examinations (/teaching/csp/PreliminaryExaminations) — Computer Science and Philosophy (/teaching/csp)**
	Preliminary Examinations (/teaching/bacompsci/PreliminaryExaminations) — Computer Science (/teaching/bacompsci)
	Preliminary Examinations (/teaching/mcs/PreliminaryExaminations) — Mathematics and Computer Science (/teaching/mcs)
Term	Hilary Term 2017 (16 lectures)

Overview

This course applies lessons that have been learnt in Functional Programming to the design of programs written in an imperative style. By studying a sequence of programming examples, each a useful software tool in its own right, students learn to construct programs in a systematic way, structuring them as a collection of modules with well-defined interfaces.

The course introduces the idea of loop invariants for understanding and reasoning about loops. The course also introduces the idea of modularising larger programs, capturing the functionality of a component of the program using an abstract mathematical specification, and describing formally the relationship between that specification and the implementation.

Through lab exercises, students learn to create, debug and maintain programs of a non-trivial but moderate size.

Learning outcomes

After studying this course, undergraduates will be able to:

1. Translate basic functional idioms into imperative ones.
2. Design simple loops, using invariants to explain why they work correctly.
3. Use subroutines and modules to structure more complex programs.
4. Specify a module as an abstract datatype, and formalise the relationship between that specification and an implementation.
5. Design simple data structures.
6. Understand the imperative implementation of some common algorithms.

Synopsis

- Basic imperative programming constructs: assignments, conditionals, procedures and loops. Comparison of imperative and functional programming. Examples.

- Method of invariants: correctness rules for **while** loops; proof of termination. Examples including summing an array, slow and fast exponentiation. Unit testing; debugging.
- Examples: string comparison, printing numbers in decimal.

- Binary search.
- Quicksort.
- Programming with abstract datatypes.
- Objects and classes as modules; specification; data abstraction.
- Reference-linked data structures: linked lists, binary trees.

Syllabus

Imperative programming constructs, with informal treatment of invariants. Procedures and modules; their use in the design of large programs; specification and implementation of abstract datatypes. Data structures: arrays, reference-linked data structures. Basic tools for program development. Case studies in design of medium-sized programs.

Reading list

There is no set text for the course, in the sense of a book that is followed by the lectures. I shall be keeping in mind a book about Oberon:

- Reiser & Wirth, Programming in Oberon, Addison--Wesley, 1992.

That book is out of print, but there is **a scanned version (http://spivey.oriel.ox.ac.uk/wiki/files/rd/ProgInOberonWR.pdf)** available.

Mike Spivey's Oberon compiler implements the Oberon-2 dialect, with a few minor differences that are explained in the Lab Manual. The language itself is described in **the Oberon--2 report (ftp://ftp.ethoberon.ethz.ch/Oberon/OberonV4/Docu/Oberon2.Report.ps)** from **ETHZ (https://www.ethz.ch/en.html)** (**local copy in PDF (http://spivey.oriel.ox.ac.uk/wiki/files/rd/o2report.pdf)**).

There are many adequate treatments of the use of logic and invariants in the development of imperative programs; one reasonably pitched one is

- Gries, The Science of Programming, Springer, 1981.

Useful additional cultural reading, recommended for reading after the course, perhaps during the Easter vacation:

- Jon Bentley, *Programming Pearls*, Dorling Kindersley, 2006.

/teaching/courses/imperativeprogramming1/index.html

Universidade de São Paulo
Brasil

Informações da Disciplina

Preparar para impressão

Júpiter - Sistema de Graduação

Instituto de Matemática e Estatística

Ciência da Computação

Disciplina: MAC0110 - Introdução à Computação
Introduction to Computer Science

Créditos Aula:	4
Créditos Trabalho:	0
Carga Horária Total:	60 h
Tipo:	Semestral
Ativação:	01/01/2003

Objetivos
Introduzir a programação de computadores através do estudo de uma linguagem algorítmica e de exercícios práticos.

Introduce computer programming by studying an algorithmic language and practical exercises.

Docente(s) Responsável(eis)

3039753 - Mauro Cesar Bernardes

Programa Resumido

A brief history of computing. Algorithms: characterization, notation, basic structures. Computers: basic unities, instructions, stored program, addressing, programs in machine language. Concepts of algorithmic languages: expressions; sequential, selective and repetitive commands; input/output; structured variables; functions. Development and documentation of programs. Samples of non-numeric processing. Extense programming and debugging practices.

Programa
Breve história da computação.
Algoritmos: caracterização, notação, estruturas básicas.
Computadores: unidades básicas, instruções, programa armazenado, endereçamento, programas em linguagem de máquina.
Conceitos de linguagens algorítmicas: expressões; comandos seqüenciais, seletivos e repetitivos; entrada/saída; variáveis estruturadas; funções.
Desenvolvimento e documentação de programas.
Exemplos de processamento não numérico.
Extensa prática de programação e depuração de programas.

A brief history of computing. Algorithms: characterization, notation, basic structures. Computers: basic unities, instructions, stored program, addressing, programs in machine language. Concepts of algorithmic languages: expressions; sequential, selective and repetitive commands; input/output; structured variables; functions. Development and documentation of programs. Samples of non-numeric processing. Extense programming and debugging practices.

Avaliação

Método

Critério

Média ponderada de provas e exercícios de programação.

Norma de Recuperação

Bibliografia

"Material didático para disciplinas de Introdução à Computação", Projeto MAC Multimídia, «http://www.ime.usp.br/~macmulti/».
V. Setzer, R. Terada, "Introdução à Computação e à Construção de Algoritmos", McGraw-Hill, 1991.
E. Roberts, "The Art and Science of C", Addison-Wesley, 1995.
H.M. Deitel, P.J. Deitel, "Como Programar em C", 2a ed., Livros Técnicos e Científicos, 1999.
J-P. Tremblay, R.B. Bunt, "Ciência dos Computadores", McGraw-Hill, 1983.
B.W. Kernighan, D.M. Ritchie, "A Linguagem de Programação C, padrão ANSI", Campus, 1990.

Clique para consultar os requisitos para MAC0110

Clique para consultar o oferecimento para MAC0110

MC102 - Algoritmos e Programação de Computadores

A partir de 2011

PRÉ-REQUISITO: não há

EMENTA

Conceitos básicos de organização de computadores. Construção de algoritmos e sua representação em pseudocódigo e linguagens de alto nível. Desenvolvimento sistemático e implementação de programas. Estruturação, depuração, testes e documentação de programas. Resolução de problemas.

Programa:

Tópicos a serem estudados (preferencialmente nesta ordem):
1 - Organização Básica de um Ambiente Computacional
2 - Variáveis, Constantes e Atribuições
3 - Entrada e Saída de Dados
4 - Expressões Aritméticas, Lógicas e Relacionais
5 - Comandos Condicionais
6 - Comandos de Repetição
7 - Vetores e Strings
8 - Matrizes
9 - Funções
10 - Escopo de Variáveis
11 - Ponteiros e Alocação Dinâmica de Vetores
12 - Algoritmos de Ordenação
13 - Algoritmos de Busca
14 - Tipos Enumerados e Registros
15 - Arquivos Textos e Binários
16 - Recursão

Bibliografia:
P. Feofiloff. Algoritmos em Linguagem C. Campus-Elsevier, 1ª. edição, 2009
H. M. Deitel, P. J. Deitel. C - Como Programar, 6ª. edição, Pearson Education, 2011.
B. W. Kernighan, D. M. Ritchie. The C Programming Language, 2ª. edição, Prentice-Hall, 1988 [Tradução: C - A Linguagem de Programação. Editora Campus, 1989]
J. L. Szwarcfiter, L. Markenzon. Estruturas de Dados e seus Algoritmos, 3ª. edição, Editora LTC, 2010
W. Celes, R. Cerqueira, J.L. Rangel. Introdução a Estruturas de Dados, 1ª. edição, Editora Campus, 2004
N. Ziviani. Projeto de Algoritmos com Implementações em Pascal e C, 3ª. edição, Editora Cengage Learning, 2011
T. Cormen, C. Leiserson, R. Rivest, C. Stein. Algoritmos - Teoria e Prática, 3ª. edição, Editora Campus, 2012
R. Sedgewick, K. Wayne. Algorithms, 4ª. edição, Addison-Wesley, 2011
A. Kelley, I. Pohl. A Book on C, 4ª. edição, Addison Wesley, 1998

Instituto de Computação :: Universidade Estadual de Campinas :: Av. Albert Einstein, 1251
Cidade Universitária, Campinas/SP - Brasil, CEP 13083-852 • Fone: [19] 3521-5838

Alisha Adam & Rohit Talreja
CS 106A - Summer 2016

CS 106A – General Information and Syllabus

Instructors

Alisha Adam
Email: aadam@stanford.edu
Office: Gates B28
Office Hours: M/W 2-4pm & by appointment

Rohit Talreja
Email: rtalreja@stanford.edu
Office: Gates B28
Office Hours: M/W 2-4pm & by appointment

Class Web Page

The web page for the class is: **`http://cs106a.stanford.edu`**. All course announcement and materials (lecture slides, handouts, assignments, etc.) will be posted to the website so make sure to check it regularly!

Course Overview

This course serves as an introduction to the fundamental techniques, programming constructs, and design strategies that form the basis of modern software. You will learn a new approach towards problem solving from a computational perspective and gain an understanding of how programming is applied across many domains. Specifically, at the end of the course, you should feel comfortable writing programs that interact with the user, manipulate graphical objects on the screen, and process and manipulate data.

Lectures and Discussion Sections

There will be four 50-minute lectures every Monday through Thursday, from 11:30am - 12:20pm in NVIDIA Auditorium. You will also attend a mandatory 50-minute discussion section each week led by one of our awesome section leaders. Section signups are handled online at **`http://cs198.stanford.edu/`**. Section signups will open on the first day of class (June 20th) and close at noon the next day (June 21st). We will send you an email with your section assignment by the evening of June 21st. Sections will begin the first week of class.

Text and Handouts

There are two textbooks for the course, both of which are available at the Stanford Bookstore. The main textbook is also available at the university library. Recommended

readings for each lecture will be posted on the lecture calendar. We will also post additional handouts and reference materials on the course website.

1) *The Art & Science of Java*. Eric Roberts (ISBN: 978-0321486127).
   This is the main textbook for the class. Starting in the second week, most of the readings will be assigned from this book. We definitely recommend having a copy for both the assignments and exams.
2) *Karel the Robot Learns Java.* Eric Roberts.
   This is a short (35-page) tutorial that introduces major concepts in programming through Karel the Robot. We will use the course reader for the first week of the course, and it will be useful for your first assignment. The reader is available electronically on the course website and in hardcopy from the Stanford Bookstore.

Note: Our exams are open-book, but you will ***not*** be able to use any digital materials (e.g. PDF version of the course reader) on the exam.

Assignments

There will be 6 programming assignments throughout the quarter, roughly one for each major topic we will cover. Assignments will require more time as the quarter progresses, so later assignments will be weighted higher (see section below regarding grading). With the exception of the last assignment, you will receive feedback on each assignment during an interactive, one-on-one session with your section leader, who will provide a rating on the following scale for both functionality and programming style:

++ *Plus-Plus:* An absolutely mind-blowing solution, the likes of which we see only a few times each quarter. If a section leader thinks an assignment is worthy of this rating, they will pass it along to the instructors for approval.

\+ *Plus:* A “perfect” submission or one that exceeds expectations for the assignment. To receive this rating, a submission often reflects additional work beyond the stated requirements, or gets the job done in a particularly elegant way.

✓+ *Check-Plus:* A submission that satisfies all the requirements for the assignment, showing solid functionality as well as style.

✓ *Check:* A submission that satisfied all the requirements for the assignment, possibly with a few small problems.

✓– *Check-Minus:* A submission that has problems serious enough to fall short of the requirements for the assignment.

– *Minus:* A submission that has extremely serious problems, but demonstrates some effort and understanding of the concepts.

−− *Minus-Minus:* A submission that shows little effort and does not represent passing work.

From previous quarters we expect most grades to be in the ✓ or ✓+ range. We have also found that using buckets for scores allows your section leader to spend more time talking about what you need to learn from the assignment without worrying about justifying each point.

After each assignment, your section leader will send out a sign-up form for an interactive grading session. We will explain the interactive grading process in more detail during the first lecture.

Assignments must be completed individually, though you are allowed to discuss high-level ideas with each other. Whenever you obtain help, you should credit those who helped you directly in the program (i.e. by adding a comment to the .java file). See the section on the Honor Code below.

Late Policy

Each assignment will be due at **5:00pm** on the day it is due, as specified on the assignment handout as well as the class calendar. Assignments must be submitted electronically as described on the course website. If you are having technical difficulties with the submission process, email your submission to your section leader **before** the deadline for the assignment.

We understand that things come up and you may need a little extra time to complete an assignment, so every student will be granted **three free "late days"** to be used during the quarter. Each late day allows you to turn in an assignment up to 24 hours late without any penalty. You can use late days individually, or combine two for a 48-hour extension on a single assignment. You may use your late days on any assignment and do not need to request permission (though it may be a good idea to email your section leader so they know that they should expect a late submission). After you have exhausted your three free late days, assignments turned in late will receive a penalty of one grade bucket per day (for example, a ✓+ will turn into a ✓, and so on). **Since this is a compressed quarter, assignments received later than 48 hours following the due date will not be graded**. This also means that you can't use all 3 free late days on the same assignment.

You can think about "late days" as pre-approved extensions to use at your discretion. As a result, getting an extension beyond the three free late days will generally not be

approved. Exceptions are made for very special circumstances, primarily extended medical problems or emergencies. **Extensions beyond the free late days can only be granted by the instructors (specifically, do not ask your section leader about extensions). To request an extension after exhausting your free late days, you must email the instructors at least 24 hours before the assignment deadline.**

LaIR Helper Hours

For most of you, this is your very first experience with computer programming. As a result, we provide extensive support and assistance throughout the quarter. Section leaders are available from 7-11pm from Sunday through Wednesday evenings in the basement of the Gates Computer Science building (room B08) to help with assignments or review course material. The latest schedule for helper hours can be found at **`http://cs198.stanford.edu`** under the "Helper Schedule" link.

Computer Facilities

The assignments in 106A will require extensive use of a computer. The preferred software is the Eclipse development environment that runs on both Mac OSX and Windows. Instructions for downloading and installing Eclipse are available on the course website. Eclipse should also be installed on all cluster and library computers across campus.

Exams

This course will have two written exams. The exams are **open-book but closed-notes**. This means that you may bring physical copies of both course texts to the exam. However, you may not use any additional typed or handwritten notes, including anything that was posted on the course website. Additionally, electronic devices are not allowed.

The midterm exam will take place **from 7pm – 9pm on Monday, July 18th (location TBD).** If you absolutely cannot make the regularly scheduled midterm, you must send a request via email to the instructors by 5:00pm on Monday, July 10th for an alternate exam time. Please include in your email all the possible times that you're available to take the exam from Monday, July 18th to Wednesday, July 20th.

The final exam is scheduled for 12:15pm to 3:15pm on August 12th (location TBD). Since this time is set by the University Registrar, **there will be no alternate time for the final exam (SCPD students see below)**. Please make sure that you can attend the final exam before enrolling in the course.

SCPD students can choose to take the exams on campus during the scheduled time or make arrangements through the SCPD office to take them remotely. In the latter case, exams should be sent back through the SCPD office by end of day PST on August 12th.

Grading

Your overall course grade will be calculated as a weighted average of the following categories:

- 50% programming assignment (weighted toward the later assignments)
- 30% final exam
- 15% midterm exam
- 5% section attendance and participation

Honor Code

Although you are allowed (and encouraged) to discuss high-level assignment ideas with other students, you should write your programs individually. Discussing assignments in such detail that another student could duplicate your code is not permitted. As mentioned earlier, when you receive help from other sources, you must credit them directly in the code by adding comments. **Any assistance used without proper citation may be considered plagiarism.**

Additionally, you are not allowed to copy code or code snippets from any sources. The only exceptions to this are the course textbook, handouts, and lecture examples, which you should cite when used. It is also a violation of the Honor Code to look at another student's code (including students from previous quarters) or show another student your code.

You may not upload your code to a public repository (such as github.com or bitbucket.com) or any other website where it can be publicly discovered. Posting code where it can be publicly discovered can be considered a violation of the Honor Code. If you would like to upload your code to a **private** repository, you may do so.

The full text of the Stanford Honor Code can be found at:
https://communitystandards.stanford.edu

Office of Accessible Education

Students who need an academic accommodation based on the impact of a disability must initiate the request with the Student Disability Resource Center (SDRC) located within the Office of Accessible Education (OAE). SDRC staff will evaluate the request with required documentation, recommend reasonable accommodations, and prepare an Accommodation Letter for faculty dated in the current quarter in which the request is being made. Students should contact the SDRC as soon as possible since timely notice is needed to coordinate accommodations. The OAE is located at 563 Salvatierra Walk, and their phone number is 650-723-1066.

VANDERBILT UNIVERSITY (http://www.vanderbilt.edu)

Scientific Computing

(/scientific_computing/)

Finding Eligible Scientific Computing Courses

On YES (https://webapp.mis.vanderbilt.edu/more/SearchClasses!input.action), to select all courses approved for credit in the Scientific Computing minor, select the "Advanced" link next to the search box, select the "Class Attributes" drop-down box on the bottom right of the advanced search page, and then select "Eligible for Scientific Computing" to find all courses.

Core Programming Courses

CS 1101 [Formerly CS 101] Programming and Problem Solving. An intensive introduction to algorithm development and problem solving on the computer. Structured problem definition, top down and modular algorithm design. Running, debugging, and testing programs. Program documentation. [3]
- or -
CS 1103 [Formerly CS 103] Introductory Programming for Engineers and Scientists. An introduction to problem solving on the computer. Intended for students other than computer science and computer engineering majors. Methods for designing programs to solve engineering and science problems. Generic programming concepts. [3]

CS 2201 [Formerly CS 201] Program Design and Data Structures. The study of elementary data structures, their associated algorithms and their application in problems; rigorous development of programming techniques and style; design and implementation of programs with multiple modules, using good data structures and good programming style. Prerequisite: CS 1101. [3]
- or -
CS 2204 [Formerly CS 204] Program Design and Data Structures for Scientific Computing. Data structures and their associated algorithms in application to computational problems in science and engineering. Time and memory complexity; dynamic memory structures; sorting and searching; advanced programming and problem-solving strategies; efficient software library use. Prerequisite: CS 1101 or 1103. [3]

Scientific Computing Courses

SC 3250 [Formerly SC 250] Scientific Computing Toolbox. Team taught course with topics illustrating use of computational tools in multiple science and engineering domains. Topics may include simulations of complex physical, biological, social, and engineering systems, optimization and evaluation of simulation models, Monte Carlo methods, scientific visualization, high performance computing, or data mining. Prerequisite: CS 1101 or 1103; Math 1200. [3]

SC 3260 [Formerly SC 260] High Performance Computing. Introduction to concepts and practice of high performance computing. Parallel computing, grid computing, GPU computing, data communication, high performance security issues, performance tuning on shared-memory-architectures. Prerequisite: CS 2201 or CS 2204. SPRING. [3]

SC 3841 [Formerly SC 293A] Directed Study in Scientific Computing. Participation in ongoing research projects under direction of a faculty sponsor. Project must combine scientific computing tools and techniques with a substantive scientific or engineering problem. Consent of both the faculty sponsor and one Director of the SC minor is required. Prerequisite: SC 3250. [1-3 each semester]

SC 3842 [Formerly SC 293B] Directed Study in Scientific Computing. Continuation of SC 3841 under the direction of the same or different faculty sponsor. Same requirements as for SC3841 [1-3 each semester]

SC 3843 [Formerly SC 293C] Directed Study in Scientific Computing. Continuation of SC 3842 under the direction of the same or different faculty sponsor. Same requirements as for SC3841. [1-3 each semester]

SC 3851 [Formerly SC 295A] Independent Study in Scientific Computing. Development of a research project by the individual student under direction of a faculty sponsor. Project must combine scientific computing tools and techniques with a substantive scientific or engineering problem. Consent of both the faculty sponsor and one Director of the SC minor is required. Prerequisite: SC 3250. [1-3 each semester]

SC 3852 [Formerly SC 295B] Independent Study in Scientific Computing. Continuation of SC 3851 under the direction of the same or different faculty sponsor. Same requirements as for SC3851. [1-3 each semester]

SC 3853 [Formerly SC 295C] Independent Study in Scientific Computing. Continuation of SC 3852 under the direction of the same or different faculty sponsor. Same requirements as for SC3851. [1-3 each semester]

SC 3890 [Formerly SC 290] Special Topics in Scientific Computing. Special topics course. [3]

These pages contain selected sections of Structure and Interpretation of Computer Programs by Harold Abelson and Gerald Sussman with Julie Sussman.

The programs are given in JavaScript, translated from Scheme by Martin Henz. The JavaScript programs can be executed (evaluated) by clicking on them.

If you are interested, here is some information on how these pages were generated, and what tools are used "under the hood".

Contents

[5 Computing with Register Machines]

UC Berkeley EECS
CS10 : The Beauty and Joy of Computing
Spring 2012

QuickLinks

UC-WISE (labs) | Piazza | bSpace
Google+ (#cs10news) | BYOB | Webcasts

Overview

CS10 , *The Beauty and Joy of Computing* , is an exciting new course offered by the UC Berkeley EECS Dept . Computing has changed the world in profound ways. It has opened up wonderful new ways for people to connect, design, research, play, create, and express themselves. However, just using a computer is only a small part of the picture. The real transformative and empowering experience comes when one learns how to program the computer, to translate ideas into code. This course will teach students how to do exactly that, using BYOB (based on Scratch), one of the friendliest programming languages ever invented. It's purely graphical, which means programming involves simply dragging blocks around, and building bigger blocks out of smaller blocks.


Fall 2009 students work together using pair programming

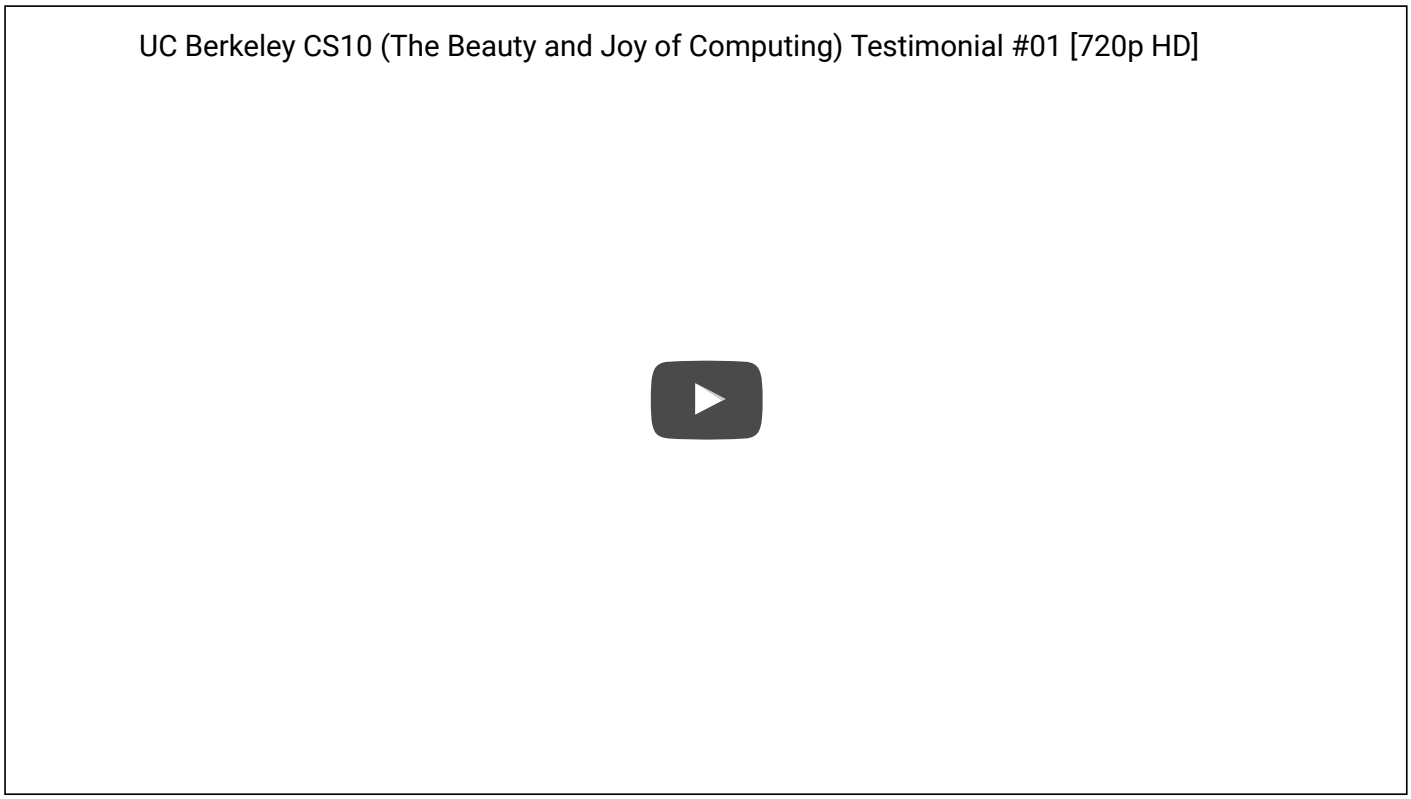
Fall 2009 students pair programming in Scratch.


Our labs are held in the Apple Orchard, which is not only the newest lab on campus with the fastest machines, but also has the most natural light!

But this course is far more than just learning to program. We'll focus on some of the "Big Ideas" of computing, such as abstraction, design, recursion, concurrency, simulations, and the limits of computation. We'll show some beautiful applications of computing that have changed the world, talk about the history of computing, and where it will go in the future. Throughout the course, relevance will be emphasized: relevance to the student and to society. As an example, the final project will be completely of the students' choosing, on a topic most interesting to them. The overarching theme is to expose students to the beauty and joy of computing. This course is designed for computing non-majors, although interested majors are certainly welcome to take the class as well! We are especially excited about bringing computing (through this course) to traditionally under-represented groups in computing, i.e., women and ethnic minorities.

Some context: in the Fall of 2009, we piloted a 2-unit version of this course as the freshman/sophomore seminar CS39N: The Beauty and Joy of Computing to 20 students. It was such a success that we decided to move ahead to make this course our new computing course for non-majors, replacing the venerable CS3L and CS3S . Last fall (2010) was a 90-person pilot and we're continuing to grow the course as word spreads to more students. We're continually replacing the weakest parts of the curriculum and hope you'll enjoy!

We will be using Pair Programming, described best by Laurie Williams, a computer science professor at North Carolina State University: "Two programmers working side-by-side, collaborating on the same design, algorithm, code or test. One programmer, the driver, has control of the keyboard/mouse and actively implements the program. The other programmer, the observer, continuously observes the work of the driver to identify tactical (syntactic, spelling, etc.) defects and also thinks strategically about the direction of the work. On demand, the two programmers can brainstorm any challenging problem. Because the two programmers periodically switch roles, they work together as equals to develop software."



18 student testimonials about CS10!

News

03-08 **Midterm Project Progress Report Submission Form**
Now available Here. Only submit ONE per group.

02-29 **Midterm Project Proposal Submission Form**
Now available Here. Only submit ONE per group.

01-29 **bSpace set up**
You should now have access to the bSpace course: "CS10 Sp12." If you don't see it, please let us know right away!

01-24 **Section #8 - it's official!**
Please see the weekly schedule below for the finalized days and times.

01-23 **Enrollment key restriction removed for UC-WISE / sage.cs / The labs**
The Spring 2012 (online) lab site was set up with an enrollment key (as many of you found out when trying to "enrol" [sic] in the course). The key has been removed, so be sure to click the "enrol [sic] me in this course" link on the laft-hand side of the page (after you log in).

01-19 **New section, first lab tomorrow (Friday the 19th at 9am)**
As the new section's (8's) first lab has already passed (Wednesday morning), it's important to go to tomorrow's (Friday's) lab at 9am if you want to attend the 8th section and/or haven't been to lab yet this week.

01-18 **Welcome to CS10, everyone!**
Those of us on staff are really excited about sharing the Beauty and Joy of Computing with you all, and are looking forward to a great semester! -The Staff

Webcasts

Webcasts of our lectures are freely available online!

- Spring 2012
- Fall 2011
- Spring 2011
- Fall 2010

Calendar

Weekly Schedule

NOTE: Discussion time for section #8 (108) is still TBD, so if you are wait-listed, please go to any of the existing Friday sections for the first week or two.

Hour	Monday	Tuesday	Wednesday	Thursday	Friday

9:00am						Lab 018 (Samir Makhani) 200 Sutardja Dai	
10:00am		Lab 014 (Navin Eluthesen) 200 Sutardja Dai	Lab 018 (Samir Makhani) 200 Sutardja Dai	Lab 014 (Navin Eluthesen) 200 Sutardja Dai	Discussion 101 (Aijia Yan) 320 SODA	Lab 018 (Samir Makhani) 200 Sutardja Dai	
11:00am		Lab 015 (Luke Segars) 200 Sutardja Dai		Lab 015 (Luke Segars) 200 Sutardja Dai	Discussion 102 (Yaniv (Rabbit)) 320 SODA		
12:00pm					Discussion 103 (Yaniv (Rabbit)) 320 SODA		
1:00pm		Lab 016 (Pierce Vollucci) 200 Sutardja Dai		Lab 016 (Pierce Vollucci) 200 Sutardja Dai	Discussion 104 (Navin Eluthesen) 320 SODA	Discussion 108 (Samir Makhani) 7 Evans	
2:00pm					Discussion 105 (Luke Segars) 320 SODA		
3:00pm	Lecture 10 Evans	Lab 017 (Navin Eluthesen) 200 Sutardja Dai	Lecture 10 Evans	Lab 017 (Navin Eluthesen) 200 Sutardja Dai	Discussion 106 (Pierce Vollucci) 320 SODA		
4:00pm	Lab 011 (Aijia Yan) 200 Sutardja Dai		Lab 011 (Aijia Yan) 200 Sutardja Dai		Discussion 107 (Navin Eluthesen) 320 SODA		
5:00pm							
6:00pm	Lab 012 (Yaniv (Rabbit)) 200 Sutardja Dai		Lab 012 (Yaniv (Rabbit)) 200 Sutardja Dai				
7:00pm							
8:00pm	Lab 013 (Yaniv (Rabbit)) 200 Sutardja Dai		Lab 013 (Yaniv (Rabbit)) 200 Sutardja Dai				
9:00pm							

Semester Schedule (subject to change)

- ***These readings are required, but are challenging - you should understand the "big idea" concepts, rather than the technical details.***
- **These readings are optional (but recommended).**

Week	Days in 2012	Readings (Sa/Su)	Lecture 1 (M)	Lab 1 (M/Tu)	Lecture 2 (W)	Lab 2 (W/Th)	Discussion (F)	HW & Projects Due
1	Jan 16 - Jan 21	• Prof. Harvey's Intro to Abstraction • Why Software is Eating the World • Is Abstraction the Key to Computing?	MLK Holiday (extended)	MLK Holiday (extended)	Abstraction	Broadcast, Animations, & Music	Welcome, Introductions, and Expectations	
2	Jan 23 - Jan 28	• Designing Games with a Purpose (GWAP) • Justices Split on Violent Games • Kinect's Future a Game Controller in Everything • Animating a Blockbuster • More readings on video games	3D Graphics	Loops and Variables	Video Games	Random, If, & Input	Anatomy of a Computer & The Power of Binary	Homework 0 Friday at 11:59pm
	Jan 30 -	• Program or Be Programmed (Video: Author			Programming			Homework 1

3	Feb 04	Speech) • Scratch: Programming for All (CACM)	Functions	BYOB	Paradigms	Lists I	Video games	Friday at 11:59pm
4	Feb 06 - Feb 11	• BtB Chapter 1 • How Algorithms Shape Our World	Guest Lecturer TA Luke Segars: Algorithms I	Lists II	Algorithms II, Order of Growth	Algorithms	Lists & Algorithms	Homework 2 Friday at 11:59pm
5	Feb 13 - Feb 18	No Reading (QUEST) **Quest Review:** Sunday, 6-9pm, 2050 VLSB	**Quest (in-class exam -- lab sections 4-8 go to 390 Hearst Mining)**	Algorithm Complexity	Guest Lecturer TA Yaniv Assaf: Concurrency	Concurrency	Algorithms & Algorithmic Complexity	
6	Feb 20 - Feb 25	• How Moore's Law Works • *Free Lunch is Over* • Spending Moore's dividend (CACM)	Presidents Day (extended)	Presidents Day (extended)	Recursion I	Recursion I	Recursion, Project Introduction, and Homework 3	Homework 3 Friday at 11:59pm
7	Feb 27 - Mar 03	• BtB Chapter 2 • Computing as Social Science	Social Implications I	Project Work	Guest Lecturer TA Aijia Yan: Recursion II	Recursion II	Recursion Revisited	
8	Mar 05 - Mar 10	• BtB Chapter 3 • BtB Chapter 4 Reading Segment 1 • BtB Chapter 4 Reading Segment 2 • Living in a Digital World	Guest Lecturer Gerald Friedland: Social Implications II	Project Work	Guest Lecturer Bjoern Hartmann: HCI	Recursion III	Social Implications of Computing	
9	Mar 12 - Mar 17	• BtB Chapter 5 Reading Segment 1 • BtB Chapter 5 Reading Segment 2 • BtB Chapter 5 Reading Segment 3 • BtB Chapter 6 (27-37) • Data Explosion Creates Revolution	Game Theory	Project Work	Guest Lecturer Raffi Krikorian: 'Twitter'	Applications That Changed The World	Midterm Prep	Midterm Project (and some general tips) Friday at 11:59pm
10	Mar 19 - Mar 24	No Reading (MIDTERM) **Midterm Review:** Sunday, 6-9pm, 2050 VLSB	Guest Lecturer Anna Rafferty: Artificial Intelligence	**Online Midterm Exam**	Guest Lecturer TA Luke Segars: Applications that Changed the World	Study for **Paper Midterm Exam, 3/22 155 Dwinelle, 6-8pm** (Info) (Histogram)	Artificial Intelligence	
11	Mar 26 - Mar 31	No Reading (Break)	Spring Break	Spring Break	Spring Break	Spring Break	Spring Break	
12	Apr 02 - Apr 07	• BtB Chapter 7 • BtB Chapter 8	Lambda + HOFs I	Lambda + HOFs I	Lambda + HOFs II	Lambda + HOFs II	HOFs & Lambdas	Blog Entry Sunday at 11:59pm
13	Apr 09 - Apr 14	• What is Cloud Computing? (Video) • *A Berkeley View of Cloud Computing*	Distributed Computing	Distributed Computing	Guest Lecturer Prof Kathy Yelick: 'Saving the World with Computing (CS + X)'	Project Work	HOFs & Lambdas Revisited	Blog Comments Friday at 11:59pm Project Proposal Friday at 11:59pm
14	Apr 16 - Apr 21	• What is IBM's Watson? • The Great Robot Race (Video) • Computers Solve Checkers -- It's a Draw • Brian Harvey's AI notes • The First	Limits of Computing	Project Work	Future of Computing	Project Work and Peer Review	Open Topic	Project Peer Review Write-up Friday at 11:59pm

		Church of Robotics • The Thinking Machine (Video) • Computer Pioneer Alan Turing • Why is Quantum Different? • Quantum Leap • Twenty Top Predictions for life 100 years from now • Apple's 1987 Knowledge Navigator (Video) • Microsoft's view of productivity in 2019 (Video) • The Future of Augmented Reality • Apple's Siri • BtB: Conclusion						
15	Apr 23 - Apr 28		Guest Lecturer Prof Armando Fox: 'Cloud Computing'	Project Work	Summary and Farewell	Online Final Exam	Final Thoughts	Final Project (and same general tips) Friday at 11:59pm
16	Apr 30 - May 05	No Reading (RRR) No Reading (Final)	RRR Week	RRR Week	RRR Week	RRR Week	RRR Week	
17	May 07 - May 12	**Final Review:** Sunday, 6-9pm, 2050 VLSB • Reading Slides • Programming Slides	Finals Week	Finals Week	**Paper Final Exam May 9th, 7-10 pm (10 Evans)** **Supplementary Handout**	Finals Week	Finals Week	

Staff

Lecturer

Dan Garcia (bio)
ddgarcia@cs.berkeley.edu
777 Soda, (510) 517-4041
OH: F 2-3pm in 777 Soda
(but please first check his Travel Schedule to be sure he's here that week!)

Teaching Assistants

Pierce Vollucci (bio)
cs10-ra@inst.eecs.berkeley.edu
OH : Th 3-4pm, Tables Outside Lab

Luke Segars (bio)
cs10-tc@inst.eecs.berkeley.edu
360 Hearst Mining
OH : W 4-5pm, Tables Outside Lab

Navin Eluthesen (bio)
cs10-ta@inst.eecs.berkeley.edu
OH : M 2-3pm, 283E Soda
& F 3-4pm, 611 Soda

Yaniv Assaf, aka **Rabbit** (bio)
rab6bit@gmail.com
OH : Th 5-6pm, 411 Soda
& F 1-2pm, 200 SD Lab

Aijia Yan (bio)
cs10-tf@inst.eecs.berkeley.edu
OH : F 11am-12pm, Tables Outside Lab

Samir Makhani (bio)
makhani@berkeley.edu
OH: F 2-3pm, 200 SD Lab

Readers

Shreya Lakhan-Pal (bio)
cs10-rd@inst

Kylan Nieh (bio)
cs10-re@inst

Head Grader Max Dougherty (bio)
cs10-rf@inst

Christian Pedersen (bio)
cs10-rc@inst

Ian Birnam (bio)
cs10-rb@inst

Grading

For the most part, we would prefer to teach this course *without* grades. What a wonderful concept, learning for learning sake! However, even though we can't change the "system" overnight, we can create grading policies that support learning as much as possible. The various course activities will contribute to your grade as follows:

Activity	Course Points	Percentage of Total Grade

Weekly Quizzes and Homework	60	15%
Paper	60	15%
Midterm Project	60	15%
Final Project	60	15%
Quest	20	5%
Midterm	60	15%
Final Exam	80	20%

How We'll Calculate Your Grade

Your letter grade will be determined by total course points, as shown in the table below. There is no curve; your grade will depend only on how well you do, not on how well everyone else does. Incomplete grades will be granted only for dire medical or personal emergencies that cause you to miss the final exam, and only if your work up to that point is satisfactory.

Points	Grade
390-400	A+
370-389	A
360-369	A-
350-359	B+
330-349	B
320-329	B-
310-319	C+
290-309	C
280-289	C-
240-279	D
< 240	F

Resources

- λ BYOB : Build Your Own Blocks
- λ Scratch Forums
- λ Blown to Bits
- λ Debugging Rules!
- λ UC Berkeley
- λ College of Engineering
- λ Department of Electrical Engineering & Computer Sciences
- λ Webcast archive of 2010Fa lectures
- λ Solutions to Lab Exercises

Contact Webmaster 2014-06-28@01:20:40 PDT

Disciplina: Ementa/Programa

FECHAR JANELA

Órgão:	CIC - Departamento de Ciência da Computação
Código:	113476
Denominação:	Algoritmos e Programação de Computadores
Nível:	Graduação
Vigência:	1971/2
Pré-req:	Disciplina sem pré-requisitos
Ementa:	Princípios fundamentais de construção de programas. Construção de algoritmos e sua representação em pseudocódigo e linguagens de alto nível. Noções de abstração. Especificação de variáveis e funções. Testes e depuração. Padrões de soluções em programação. Noções de programação estruturada. Identificadores e tipos. Operadores e expressões. Estruturas de controle: condicional e repetição. Entrada e saída de dados. Estruturas de dados estáticas: agregados homogêneos e heterogêneos. Iteração e recursão. Noções de análise de custo e complexidade. Desenvolvimento sistemático e implementação de programas. Estruturação, depuração, testes e documentação de programas. Resolução de problemas. Aplicações em casos reais e questões ambientais.
Bibliografia:	Básica Cormen, T. et al., Algoritmos: Teoria e Prática. 3a ed., Elsevier - Campus, Rio de Janeiro, 2012 Ziviani, N., Projeto de Algoritmos com implementação em Pascal e C, 3a ed., Cengage Learning, 2010. Felleisen, M. et al., How to design programs: an introduction to computing and programming, MIT Press, EUA, 2001. Complementar Evans, D., Introduction to Computing: explorations in Language, Logic, and Machi nes, CreatSpace, 2011. Harel, D., Algorithmics: the spirit of computing, Addison-Wesley, 1978. Manber, U., Introduction to algorithms: a creative approach, Addison-Wesley, 1989. Kernighan, Brian W; Ritchie, Dennis M.,. C, a linguagem de programacao: Padrao ansi. Rio de janeiro: Campus Farrer, Harry. Programação estruturada de computadores: algoritmos estruturados. Rio de Janeiro: Guanabara Dois, 2002.

CS 105L: Introduction to Computer Programming using JavaScript Spring 2016

Department of Computer Science
University of New Mexico

Instructor:

Joel Castellanos
Department of Computer Science

Course Description

CS-105L, Introduction to Computer Programming, is a gentle and fun introduction. Students will use a simple Integrated Development Environment to author small programs in a high level language that do interesting things.

Pre-Requisites: None.
Co-Requisites: None.

This semester, we will be programming in JavaScript to make small, but fun programs that run in Web Browsers. In addition toprogramming, the course also covers some of the great ideas in computer science such as modeling, visualization, emergence, search engine page ranking systems, and complex adaptive systems. Throughout the course, students will author many short programs that preform two-dimensional graphics, animations, customized image manipulations and some simple games.

CS-105 is designed as a first course in computer programming for:

1. Pre-CS majors who do not have previous programming experience and are not yet ready for the fast pace and rigor of CS-152 (Computer Programming Fundamentals).
2. Students without programming experience who want to learn the basics of programing, an introduction to the JavaScript, HTML 5, and CSS to gain practical skills in Web design, how to create customized multi-media effects and other tasks.

Syllabus & Lab Schedule:

CS-105 Syllabus for Spring 2016

CS-105 Schedule of lab, office hours and tutor times and e-mails.

Textbooks and Web Resources:

Eloquent JavaScript, 2nd Edition
by Marijn Haverbeke
ISBN-13: 978-1593275846

w3schools.com: HTML 5 Tutorial

w3schools.com: JavaScript Tutorial

w3schools.com: CSS Tutorial

Software:

WebStorm: JavaScript Integrated Development Environment (IDE) Note: with your UNM e-mail, you can get a free student license.

putty.exe Windows client for remote terminal access to linux.unm.edu.

Lecture Notes, Videos, Labs and Source Code

Week 1:

1. Video: Khan Academy: What is Programming?
2. Video: Khan Academy: Learning Programming on Khan Academy
3. Video: Khan Academy: Drawing Basics
4. Notes: Welcome to CS-105
5. Lab: Lab 1: I Love to Draw
6. Code: CS105_Lab1_MyFirstName_MyLastName.html This is the Template HTML file I showed in class.

Week 2:

1. Video: Khan Academy: Coloring
2. Video: Khan Academy: Variables
3. Video: Khan Academy: Animation Basics
4. Read: w3Schools on if statements (and check out the "Try it Yourself" button).
5. Notes: Variables and the Draw Loop
6. Code: CS105_Lab2_BouncingBall.html This is the starting bouncing ball code for lab 2. Start with this and change it.
7. Code: processing.min.js JavaScript file NEEDED by to include `Processing` code in your JavaScript program.
8. Lab: Lab 2: Bumper Ball

Week 3:

1. Video: Khan Academy: Functions
2. Video: Khan Academy: Logic and if Statements
3. code: Lab 2 Solution by Zach Ward Zach has portals! - he also some ellipses collisions - not perfect, but what he has works better than the unmodified circle collision.
4. code: Lab 2 Solution by Alex Booher A simple Pong game with a ***very simple*** AI - but since an interactive game and AI were not required, this is extra credit.

Week 4 & 5:

1. Read: Eloquent JavaScript, Chapter 1: Values, Types, and Operators
2. Video: Khan Academy: Interactive Programs: Responding to Mouse clicks and Mouse movements.
3. Video: Khan Academy: Resizing with Variables
4. Lab: Lab 3: Bumper Pool: Using mouse, collisions and physics Part 1: Board setup and cue ball shot.
5. Code: processing.min.js JavaScript file NEEDED by to include `Processing` code in the same folder as your JavaScript program.
6. Code: CS105_Lab3_MouseDrag.html
7. Code: CS105_Lab3_ChangeInTime.html
8. Code: CS105_Lab3_DragBall.html This is a particial solution to lab 3. It is the code we developed in class on Tuesday, Feb 16. It combines the mouse drag with the change in time code. As we left it in class, it sets the X speed after click-and-drag, but does not yet set the y speed. To restart the ball on the right edge, reload the page.

Week 6:

1. Video: Khan Academy: Looping
2. Video: Khan Academy: Intro to Arrays
3. Read: Eloquent JavaScript, Chapter 2: Program Structure
4. Lab: Lab 4: Bumper Pool: Using mouse, collisions and physics Part 2: Wall collision, friction, and speed vector.
5. Code: processing.min.js JavaScript file NEEDED by to include `Processing` code in the same folder as your JavaScript program.
6. Code: CS105_Lab3_Pool_Part1_Joel.html
7. Notes: Loops
8. Code: Factors.html Example how to get input from a web page and a while loop inside the draw loop.
9. Code: PrimeFactors.html Modification of Factors.html that displays the input number's prime factors.
10. Code: Lines1_LinesInALoop.html Drawing Lines Example 1: Use a loop to draw many lines.
11. Code: Lines2_DrawingAndErasingLines.html Drawing Lines Example 2: Cycles between drawing and erasing lines.
12. Code: Lines3_CrossingSetsOfLines.html Drawing Lines Example 3: Draws a crossing set of lines in different colors. Then erases the lines and starts over.
13. Code: Lines4_100sOfColors.html Drawing Lines Example 4: Drawing lines smooth color changes.
14. Code: Lines5_StringArt_SingleCorner.html Drawing Lines Example 5: Line art that draws straight lines that have the illusion of curving.

15. Code: Lines6_StringArt_FourCorners.html Drawing Lines Example 6: Cool line art that draws 8 lines each frame.
16. Lab: Lab 5: Line Art
17. Code: LineArt_Anjuli_Kvamme.html Lab 5 Line Art solution by Anjuli Kvamme

Week 7 & 8:

1. Midterm exam on Thursday, March 10.
2. Lab: Lab 6:Publishing a Webpage.
3. Code: Lines7_StringArt_Slider.html Example JavaScript animation that responds to slider movements.
4. Notes: Midterm Exam Review

Week 9:

1. Exam: Midterm Exam: 2016 Spring
2. Code: MidtermExam_2016_Spring.html Exam questions in a single program.
3. Code: ColorChooser.html Example showing 3 sliders used to set the Red, Green and Blue value of the canvas background.
4. Code: CS105_Lab7_BouncingBalls.html This sample program combines much of what we did in the past to draw two balls that move with friction. They bounce off the walls and off each other. Also, moving the mouse over a ball, gives the ball a kick. This is a great starting place for lab 7. All you need to do is to add 98 balls and you are done!
5. Lab: Lab 7:A Hundred Bouncing Balls on the Web.
6. Code: http://www.unm.edu/~botelloc/home.html Cin's Milkyway: JavaScript Website by Cindy Botello.
7. Code: http://www.unm.edu/~boohera/home.html Art from Music: JavaScript Website by Alexander Booher.

Week 10:

1. Code: WalkAndTurn.html Starting point for lab 8.
2. Notes: Computer Tour

Week 11 & 12:

1. Code: CryptogramGame.html Starting point for lab 9.
2. Lab: Cryptogram Game: Requirements and Algorithms
3. w3schools: Creating a drop-down menu using the HTML <select> Tag
4. w3schools: How to get the index and value selected in a drop-down menu
5. Code: ElasticCollision.html Elastic Collision Example

Week 13 & 14:

1. Web: Khan Academy: Index of all Khan Acamemty's Computer Programming Courses
2. Web: Khan Academy: Intro to HTML/CSS: Making Web Pages. Text emphasis, HTML Images, HTML Tabels, CSS font familiew and font sizes HTML connection to JavaScript (Selecting by id),
3. Web: Khan Academy: Advanced JS: Games & Visualizations. Buttons, 3D Shapes, Making a Memory Game, Objects, Object Constructors, Object member fields (this.) and Object member functions (.prototype.)
4. Code: MemoryGame.html This version has a getImage(filename) function that looks in the current directory for the file and it also has a getKhanImage(filename) function that gets the image form the Khan Academy webpage.
5. image: ladybug.png

Week 15:

1. Code: Flapper.html Image Rotation Example: Flapper.
2. image: flipperLeft.png