I. Overview

Computer Science 1 is a broad introduction to some of the most important concepts and “Great Ideas” in the field of computer science. To paraphrase Alan Biermann, this course is for people who sometimes wonder

“… what is meant by a Java class, a file server, ethernet, TCP/IP, or Pretty Good Privacy … You have been told you need a compiler, but why do you need it? What does it actually do? Watch out for certain types of problems — they are ‘NP-complete,’ and you may not be able to compute the answer. You want to have a neat graphics picture jump onto your Web page when you push the button labeled ‘surprise.’ Can you make it happen? Your laptop runs at 2.5 gigahertz: that is two and one-half billion what per second? (And, by the way, what exactly is a computer?)”

Because this course is designed for students with no previous background in computing, it will not be adequate preparation for many of the advanced computer science courses at Harvard; however, students who successfully complete Computer Science 1 can certainly move on to CS-50 next year. CS-1 may not be taken for credit after completing CS-50, however.

This course will expose students to

1. The principles and practices of object-oriented programming (OOP) using a methodology that places a high value on programs that not only generate the “right answers,” but that are also easy to read, maintain, and modify. These elements are important in any programming, regardless of the specific language used.

---

1 Excerpted from page xiii (the Preface) in Great Ideas in Computer Science with Java, by Alan W. Biermann and Dietolf Ramm. Published by the MIT Press. ISBN: 9780262024976
Much of the programming will be done using **Java**, through which we will investigate fundamental data structures, and their algorithms.

Preliminary programming exercises will utilize HTML and the **Scratch** environment, free software that runs on both Mac OS and Windows OS platforms.

The landscape of computer science as it exists today, with some reference to its past and future. This will enable us to touch on a variety of really fascinating topics and intellectual paradigms, i.e., some of the “Great Ideas in Computer Science,” such as

- **Simulation**: Suppose you wish to observe something, but you cannot because it’s too expensive or too slow or just impossible to observe. You might be able to get your wish if you can successfully simulate that “thing” (e.g., a physical process or experiment). This means you must discover a model for that thing, program it, run the program and do your observations. If your model is good, you will be able to see what would have happened with the original thing, had you been able to observe it.

- **Computer Architecture**: When you purchased your last personal computer, what did you actually get? In fact, you received hardware that executes machine language instructions in the fetch-execute cycle at a very high speed, some memory, a hard drive, and some input-output devices (like a keyboard and display). You will learn what machine instructions are, how the fetch-execute cycle works, and how memory is used to enable the machine to do its job. You will learn the operation and mechanisms of the bare-bones computer and we’ll demystify a lot of computer jargon that you might not otherwise understand.

- **Language Translation**: But how can we type Java programs or programs in other languages into our computer and expect them to work, given that the basic computer hardware can only understand very primitive instructions? The solution is to build a translator that will transform the language that we prefer into a form that the machine can process. You will learn what people mean when they say they “compiled” a program, and you will do some compiling yourself.

- **Security and Privacy**: While computers and networking are critically

---

2 A few of these summaries are paraphrased from Biermann and Ramm’s descriptions.
important in our lives, they also can become vehicles of mischief. What if our personal secrets, our medical information, and/or our financial records are stolen and sold online? What if we become dependent on machines and suddenly they cease to work because of an electronic attack? We will explain the kinds of attacks that might be brought against individuals or organizations. And we’ll describe some of the defenses that one can use, including various encryption methods.

- **Computer Communications**: You will learn what a Local Area Network (LAN) is, and how packets get shipped around. You will learn also about Wide Area Networks, especially the Internet, what they are and how they work. You will learn some networking terminology (UDP, TCP/IP) and you will learn a bit about addressing schemes, network servers, and what the “Internet of Things” is all about.

- **Low-level Machine Architecture**: Out of very simple logic gates, such as the AND, OR and NOT, we can easily create circuits that compute primitive and complex functions. It’s also interesting to understand how such circuits are embodied by transistors and very large scale integration (VLSI) chips.

- **Program Execution Time**: We will discuss the limitations of computer science, describing a major hitch that prevents scientists from solving some important problems. It turns out that it may take a billion years of computation to solve certain problems, so we just cannot hope to do them. Even if machines get much faster, these problems will continue to be out of reach. We will introduce the ideas of tractability, referring to problems that usually can be solved in practical situations, and intractability, referring to problems that tend to require too much execution time to solve. We’ll provide you with examples of both kinds of problems so that you may gain some intuition for these phenomena.

- **Noncomputability**: There is a class of very strange problems that mathematicians have proven can never be solved by any computer within known computational paradigms. This mystical and elusive class of programs seems to place an impenetrable blockage to progress in certain aspects of computer science.

---

3 See, for example: [http://www.informationisbeautiful.net/visualizations/worlds-biggest-data-breaches-hacks/](http://www.informationisbeautiful.net/visualizations/worlds-biggest-data-breaches-hacks/)
3 Additional topics, as time allows, such as:

- **Parallel Computation:** If a problem requires too much time to solve with one computer, perhaps we could spread it across many machines, thousands or millions of them, and then solve the very time-consuming calculation. We investigate this idea with both positive and negative results. Though many computations can be speeded up tremendously by putting them on parallel machines, one cannot always do this easily and sometimes the speedup is not enough.

- **Computer Graphics:** How do computers render and manipulate photographs, drawings and movies — some of which may involve pictures of things that do not exist (or perhaps could never exist)? Why is human face recognition a difficult computational problem for computers (not for humans), and what sort of 3D modeling techniques are being developed for face modeling and animation?

- **Artificial Intelligence:** Theories of machine intelligence have evolved over several decades; we can examine the idea of representing knowledge and what it means to use that knowledge to “understand,” as well as the possibility of automating the learning process. We can also study problem-solving and how knowledge contributes to the ability to solve problems. In addition, we can examine how to use this theory to build machines that seem to be intelligent. Some examples are a human speech-recognition system, a game-playing system, and a so-called expert system.

- **Intellectual Property and Digital Information:** Copyright infringement of digital data flourishes (think of MP3’s), assisted in large part by advancements in certain information technologies. Unfortunately, the potential for this problem to worsen is great, as it is not very difficult to defeat digital rights management (DRM) systems; this presents a nightmare to book authors, musicians, publishers and others, who want the protection of copyright law to balance private interest with the greater public good. It’s important to understand what sorts of laws exist that define and protect intellectual property for hardware and software-based inventions.
II. Staff

Faculty: Dr. Henry Leitner
51 Brattle St., rm. W-719
(617) 495-9096
email: leitner@harvard.edu

Dr. Leitner will generally be available for consultation on Tuesday and Thursday morning (right after class) in Maxwell-Dworkin room G-115. He can usually be reached at his Brattle St. office on Wednesday mornings, from approximately 9:30 AM until 10:30 AM, but it’s probably best to arrange a definite appointment by phone or email.

Teaching Fellows: Mr. Dimitri Kountourogianni (head TF), dimitrik@alum.mit.edu
Mr. Filip Bujaroski, fbujaroski@college.harvard.edu
Ms. Danielle Feffer, dfeffer@college.harvard.edu
Mr. Nicholas Mahlangu, nmahlangu@college.harvard.edu
Ms. Frances S. Shapiro, fshapiro@college.harvard.edu
Ms. Maria Zlatkova, zlatkova@college.harvard.edu

others, if enrollment warrants

The TFs are responsible for grading homework and for helping students, in general, with the material covered in this course. The head TF will assist with a number of administrative matters, such as maintaining the course website.

Each student is expected to attend a mandatory 60-minute section meeting every week; these section meetings are run entirely by the teaching fellows, and will probably be held in Science Center classrooms on Mondays, though there might be a Sunday evening section. The precise times and places will be announced later on.
III. What to Read

You should acquire the habit of consulting our course website often (perhaps once every couple of days). The URL is:

http://www.fas.harvard.edu/~cs1

Optional reading materials for this course are for sale at the Harvard COOP bookstore. They will be on reserve also at the Cabot Science Library in the Science Center or at the Gordon McKay Library in Pierce Hall.


Although we do not recommend that you acquire any of the following books at this time, some of these texts may be of interest to those of you who wish to explore specific topics in more depth than we will have time for in class.

Computer Science, An Overview (12th edition), by J. Glenn Brookshear. Published by Pearson, 2014. The ISBN # is 978-0133760064. This book offers a clear and concise survey of computer science, covering a wealth of topics and presenting the scope of the discipline as well as the terminology in the field.


The Linux Command Line: A complete Introduction, by William Shotts. Published by No Starch Press, San Francisco, 2012. The ISBN # is 9781593273897. Since you will be writing your Java programs in a Linux environment, you may find it helpful to learn a bit more about this important operating system.

IV. The Problem Sets, the Term Project, and Grading

The majority of the assignments will involve problem solving using a Mac OS or Windows-based personal computer. Some of the homework exercises will be of the short “paper-and-pencil” variety. You will be required to do “electronic submission,” a process we will describe in class.

Please do not attempt to finish up an assignment during lecture or during section! Problem sets will be due, in general, prior to 5:00 PM on Fridays.

Each student is allotted a budget of 5 no-penalty “late days” for use throughout the term. Additional late days may be “purchased” for a 6-point penalty each. Extra no-penalty late days may ordinarily be obtained only by submitting a doctor’s note to Dr. Leitner. Exceptions to these rules may be sought for unusual circumstances by
petition. Do not fritter away the no-penalty late days early in the semester; you may find yourself wishing you had saved a few toward the end of the semester. If **electronic submission of your work is more than 10 minutes late arriving, then it will be considered a full day late.** At section meetings and during regular office hours, your assigned TA will return your graded homework to you. As I usually plead, please, *please*

**Don't fall behind on your problem sets!**

Just as you cannot expect to learn how to drive a car by reading about it or by watching other people do it, the same holds true for working with computer hardware and software.

Get started on the problem sets early — this is one course you simply cannot “cram” for at the last minute, so don’t even try! We cannot stress this strongly enough. Remember that some of you may find the problem sets to be time-consuming on occasion, so please take your other commitments into consideration.

Student final grades in Computer Science 1 will be based on

- the homework (60%),
- one open-book exam (25%),
- a term project (15%), and
- your teaching fellows’ appraisal of individual achievement *(priceless).*

**For the term project, you will have two distinct alternatives:**

- An original paper that is presented in the form of a **website**, covering in depth, a “Great Idea” in computer science. The subject matter does not have to be one of the topics we have explicitly touched on in class this year. A colleague of mine at Stanford named Eric Roberts has been teaching a course named “The Intellectual Excitement of Computer Science” for several years; on our course website I will post links to some of the better projects completed by both CSI students as well as students in Prof. Roberts’ course over the past few years. If you are interested in this option, your project need not have animations or flashy graphics, but we DO expect the website to be well-organized and to present your “paper” in a form that’s easy to navigate.

- The alternative option is to write an original program written in **Java** that will
run on the nice.harvard.edu system using javac.\(^5\) We will post some specific examples of Java-based projects on our website later on.

A document describing the term project in more detail (along with specific suggestions) will be handed out later in the semester.

V. **A Commentary On Programming Ethics\(^6\)**

A computer program written to satisfy a course requirement is, like a paper, expected to be the original work of the student submitting it. Copying a program from another student or from any other source is a form of academic dishonesty, as is deriving a program substantially from the work of another.

Persons who do not know how to program a computer are understandably puzzled about how the concept of plagiarism could possibly be applicable in a computer course. Is a program not an exact object, like a number, and must not any two correct solutions to a programming problem be identical? The truth is quite different. Superficial appearances aside, computer programs more closely resemble essays than numbers. The copyright laws recognize this; so do standards of behavior at Harvard.

Two programmers may adopt radically different approaches to solving the same problem, as different as the ideas of two students asked to write critically on the same painting. Very small programs do not admit this much variability in overall design, but anything over a page long certainly does, unless the design itself was specified as part of the exercise. But even when two programs are based on the same overall design, the variation in possible form of expression is vast.

Programming courses attempt to teach graceful and forceful forms of expressions for computational ideas, but every programmer has idiosyncrasies of style and vocabulary. The likelihood of two programmers independently creating identical programs of more than two or three lines in length is no larger than the likelihood of two writers independently writing identical paragraphs. And programs that are

\(^5\) You can also use an integrated development environment for Java on your personal computer, such as *Eclipse* or *NetBeans*. Just be sure to test and submit your programs on the nice.harvard.edu Linux system.

\(^6\) The principal author of this section is Prof. Harry R. Lewis, Harvard College Professor of Computer Science at Harvard.
identical except for their choice of names (for example, one has “x” everywhere the other has “y”) are as improbable as two short stories that are identical except for the names of the characters. Paraphrase is as possible, and as dishonest, with programs as with papers; two programs can be copies of each other even though no single line of one is identical to any line of the other. There should be little confusion about what is legitimate and what is not in the production of a computer program; the rule is simple, simpler than in expository writing, since programming generally does not involve library research and use of sources: **Do not submit as your own work a program based on the work of another! Violations of this rule is plagiarism; it is dishonest behavior, and the penalty for it is requirement to withdraw from Harvard College.**

Two obvious “exceptions” to this rule may be noted in passing. Courses sometimes supply the main idea or even some of the text of a program that is to be completed as an exercise; naturally, students are expected to use this assistance. And there is merit in “copying from oneself” in a course that develops cumulative programming skills. Here programs differ from papers; no author would want to write two different pieces with several paragraphs in common, but with computer programs, this is not unusual. A skill taught in programming courses is how NOT to reinvent the wheel; when a small phrase or short sentence has proven useful and reliable in one program, a programmer should feel free to reuse it if the same thing needs to be said in another program. Such clauses play the role of aphorisms; they make a point but they are not the main point of the piece being written.

Of course, neither of these examples obscures the basic point that a program submitted as original work should not have been derived from the work of another unless the course has specifically permitted this.

How much help on a programming exercise may you obtain before you are stealing, rather than being assisted? Teaching fellows and user assistants know the limitations of what is fair and legitimate; their goal is to help you understand how to solve your own problem, not to solve it for you. If you seek help from other students you are treading on much thinner ice. When a student answers a simple factual question which could have been answered out of a manual, no violation of principle is involved; it is not dishonest to ask another student the value of $\pi$ or the statement of the Pythagorean Theorem. But the more your request is for part or all of the solutions to the programming exercise itself, rather than for general factual information, the less acceptable it is. In the extreme case one student asks for and receives the actual text of a program which both were to have created independently; in this case both are guilty of academic dishonesty.

**In the Harvard College Handbook for Students** is a section related to collaboration:

*It is expected that all homework assignments, projects, lab reports, papers, theses, and examinations and any other work submitted for academic credit will be the student’s own. Students should always take great care to distinguish their own ideas and knowledge from information derived*
from sources.

In some courses students are expected to work in teams on the implementation of very large programs. Just because you see two students huddled over the same terminal and discussing programs in great detail, do not assume that this is standard and acceptable behavior in your course! If you have any doubt about what type of collaboration is permissible, do not make assumptions: ask the instructor. A general argument that you were only doing what you saw others doing is not a legitimate defense.

VII. Syllabus

The following calendar and outline should give you an idea of how CS-1 will progress.

<table>
<thead>
<tr>
<th>Date</th>
<th>Topics to be Covered and Optional Reading</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tues., Jan. 26</td>
<td>Course overview and a brief look at some of the “great ideas,” including the notion of algorithm and its relationship to programming.</td>
</tr>
<tr>
<td>Thrs., Jan. 28</td>
<td>Introduction to programming in <em>Scratch</em>, a visual, object-oriented programming language.</td>
</tr>
<tr>
<td>Tues., Feb. 2</td>
<td><em>Scratch</em> programming, conclusion</td>
</tr>
<tr>
<td>Tues., Feb. 9</td>
<td>Elementary programming in Java (variables, constants, assignment, console output). Comparison with <em>Scratch</em> constructs.</td>
</tr>
<tr>
<td>Thrs., Feb. 11</td>
<td>Elementary Java, part 2: arithmetic operators, strings, parameterless methods, nested loops using <strong>for</strong>.</td>
</tr>
<tr>
<td>Tues., Feb. 16</td>
<td>Elementary Java, part 3: Boolean operators and expressions; conditional evaluation using if-else; operator precedence.</td>
</tr>
<tr>
<td>Thrs., Feb. 18</td>
<td>Keyboard input using the <em>Scanner</em> class; formatted output using <em>printf</em>. Integer overflow and floating-point imprecision.</td>
</tr>
<tr>
<td>Tues., Feb. 23</td>
<td>Printing interesting patterns with nested loops. A brief look at</td>
</tr>
</tbody>
</table>
Java graphics.


Tues., Mar. 1  Simulation and Monte-Carlo methods through the use of pseudo-random numbers. A somewhat whimsical look at a particular aspect of computing history.

Thrs., Mar. 3  ASCII and UNICODE conventions for character encoding. Fun with String objects.

Tues., Mar. 8  Object-oriented programming in Java. Constructor methods; setters and getters.

Thrs., Mar. 10  Information hiding and encapsulation through the use of public and private access control.

Mar. 12 — 20  SPRING RECESS!


Thrs., Mar. 24  Illustrating a theorem in geometry using objects and arrays. Two-dimensional arrays. A program that learns through experience.


Thrs., Mar. 31  Introduction to file I/O. Cryptography — the science of disguising information by using various “ciphers” and other methods for encoding and decoding.


Thrs., Apr. 7  Low-level hardware: switching circuits, boolean logic and transistors. Combining AND, OR and NOT gates to create useful circuits. Designing memory devices and simple arithmetic circuits.
OPEN-BOOK EXAM TODAY! Covers Java programming up through March 31 lecture.

Computer architecture: The P88 and typical von-Neumann architecture. CPU, memory, bus, peripherals, storage, etc. The basic machine cycle (fetch, decode, execute, store) and P88 instruction set (including binary representation). Viewing programs as data and data as programs.

Formal languages and grammars. Compilation and the assembly processes — lexical analysis, parsing, code generation.

What can and cannot be computed? What are “tractable” and “intractable” computations? Introduction to functional programming and symbolic computation.

Guest lecture on applications of computing in the area of graphics (including animation, virtual reality and the modeling of human faces). This is the last regular class meeting.

No class! Reading period begins today and lasts up through May 4.

Term projects are due today prior to 5 pm.

"First, they do an on-line search."