Project Number: IQP-CK-IB02-51

AN INTRODUCTORY CS COURSE FOR NON-MAJORS

An Interactive Qualifying Project Report

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Bachelor of Science

by

Brian J Corcoran

Brian J. Corcoran

and

Bradley K. Noyes

Date: May 2, 2003

Approved:

Professor Carolann Koleci, Major Advisor

Glynis M. Hamel

Professor Glynis Hamel, Co-Advisor

1. **education**
2. **teaching**
3. **computers**

Abstract

Our project evaluates the current state of computer science and programming education for non-computer science majors. Interviews and surveys were conducted to find out what concepts and techniques WPI students and professors are looking for in a computer science course for non-computer science majors. We propose a new computer science course for non-majors. Significant components of the proposal include using projects based on students' majors and teaching using the Python programming language.

Acknowledgments

We would like to express our gratitude to our advisors, Professor Carolann Koleci and Professor Glynis Hamel, who helped us throughout the whole project. We would also like to thank the students and professors who filled out our surveys. In particular, we would like to thank the following WPI professors for allowing us to interview them: P.K. Aravind, Terri Camesano, John Goulet, Judith Miller, Creighton Peet, and Mike Ciaraldi.

We would like to express additional thanks to John Goulet, whose Linear Algebra course served as an inspiration for this project.

We would also like to thank Mr. Jeffrey Elkner, co-author of the book *How To Think Like A Computer Scientist*, for allowing us to interview him about his experiences teaching Python.

Finally, we would like to thank Professor Joe Wong and his CS 1001 class for testing our major-based projects.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

"Why another computer-science course?" WPI already has many computer-science (CS) courses, and its CS program is well respected. However, only one course is targeted for non-computer-science majors, CS 1001. Deficiencies in CS 1001 have resulted in a proposal by the CS department to remove it from the curriculum. The goal of our IQP is to create a computer-science course focused on the needs of those WPI students who are not majoring in computer-science (non-majors [1]).

We chose this topic for a multitude of reasons. We believe that computer science introduces many fundamental concepts which are important in all areas of study; concepts such as abstraction, debugging, algorithms, and logical thinking. Furthermore, the ability to write programs is becoming increasingly important in all science and engineering disciplines. In particular, the ability to use programming as a tool fits perfectly into WPI's goal of creating well-rounded graduates who are familiar with science, humanities, and technology.

In choosing our IQP, we hoped to address an issue where we could make a real impact. We believe that this report addresses many of the problems with the

[1] For the purposes of this report, we do not consider electrical and computer engineering majors or students minoring in computer science as "non-majors"

current situation, and we feel that our proposal would greatly improve the quality of non-major CS education if it were adopted.

As CS majors, we feel programming is an excellent tool for many different disciplines. In many cases, programming can be as useful as general mathematics is to the scientist or engineer. However, we feel that many non-majors do not realize the benefits of programming. Our hypothesis in proposing this IQP is that students do not take programming courses because they do not feel the courses are applicable to their major field of study.

We surveyed undergraduate students at WPI in an attempt to determine why they take or avoid CS courses. With the the survey results, we then designed a programming course aimed at increasing non-major student interest. In addition to the student survey, we interviewed and surveyed WPI faculty and professionals in the science and engineering communities to determine what they feel is necessary for a non-major to learn in an introductory computer-science course.

Our final proposal consists of a programming course which we believe non-majors will find interesting and will help them excel in their field of study.

Chapter 2

The Problem

I was kinda interested at first, but after all the horror stories I've heard?

–WPI non-major, regarding introductory CS courses at WPI

2.1 Introduction

We believe that WPI's current CS courses do not meet the needs of non-majors. To show this, we first examine the current CS courses which are taken by non-majors. We then take a closer look at those courses, and identify deficiencies based on student surveys, faculty interviews, and our own knowledge. We define the key problems which face any course for non-majors. Finally, we establish criteria for evaluating solutions to these problems.

2.2 Many WPI Non-Majors Do Not Take CS

Enrollment among non-majors in the introductory programming courses at WPI is low. While 70% of the non-majors we surveyed indicated that they believe non-

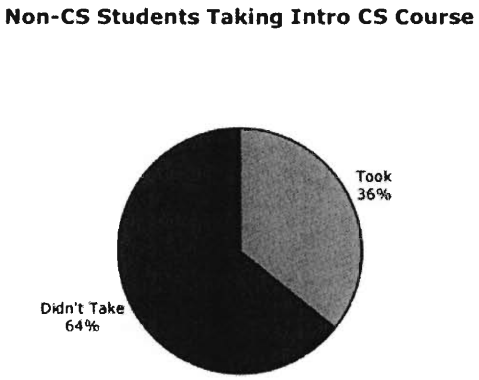


Figure 2.1: Non-majors who have taken introductory CS

majors should learn programming, only 36% of non-majors had taken a programming course (see Fig. 2.1). To determine why enrollment is so low, we examine the different introductory CS options for non-majors.

2.2.1 Current WPI introductory CS courses

WPI offers three 1000-level, or freshman-level, CS courses. These are:

CS 1001 Introduction to Computers

CS 1005 Introduction to Programming

CS 1006 Object-Oriented Introduction to Programming

The recommendation for CS majors is that they begin with either CS 1005, taught in C++, or CS 1006, taught in Java. Neither course counts toward the six units [1] of CS required for a computer science degree. The recommendation for non-majors is to take at least one of CS 1005, CS 1006, or CS 1001. CS 1001 is taught in FORTRAN. The course descriptions for all three courses state that no prior programming experience is required.

[1] At WPI 1/3 unit is equivalent to 3 credit-hours at most other schools.

2.2.2 Problems with CS 1005 and CS 1006

I've heard that the intro CS courses (i.e. CS 1005) are much too hard if you come to WPI to learn programming and haven't learned it before.

–WPI non-major

Out of the three courses which non-majors tend to take to learn programming, CS 1005 and CS 1006 are more intensive and are geared toward the student who is majoring, minoring, or pursuing a concentration in computer science. However, most non-majors end up taking CS 1005 or CS 1006 and have a difficult time with the courses. In a survey of students we conducted, 38% of non-majors said that the introductory CS courses at WPI were difficult or very difficult. [2] CS 1005 and CS 1006 were originally meant for students who do not have prior programming knowledge. [18] Over the years, however, CS 1005 and CS 1006 have developed into courses which do require prior programming knowledge. [17] As a result, students who take CS 1005 and CS 1006 to learn programming with no prior knowledge do poorly in the courses and end up receiving NRs (WPI equivalent of failing). Our survey results show that this difficulty has earned the courses a negative reputation among students. Some comments we received from non-majors are:

> *I was kinda interested at first, but after all the horror stories I've heard?*
>
> *I have not taken a CS class myself, but I have heard from other students that they are difficult if you have never taken programming before.*
>
> *I know nothing about CS courses except they are considered hard.*
>
> *I haven't taken any CS courses myself, but I've heard from non-CS majors who've taken introductory CS classes that they are very difficult if*

[2] See Appendix A.1.2 for survey results.

> *you have no prior programming knowledge.*

> *I am not sure because I haven't taken it but my roommate says the classes are very hard and she is non-CS.*

We believe this negative reputation is causing a low turnout of non-majors for CS courses.

Since CS 1005 and CS 1006 are meant to prepare CS majors for a career in programming, the focus and the concepts being taught in the courses are computer-science centric. The focus on computer science concepts and theory and the increasing difficulty of the courses tend to discourage non-majors. Of the non-majors we surveyed, 22% indicated that the current introductory CS courses covered too much theory. Additionally, 49% of non-majors reported that they did not learn how to program in their introductory CS course.

Our survey results indicate that non-majors want a more practical programming experience to apply to their major field of study. Forty-seven percent of non-majors reported that they did not plan on using the knowledge from their introductory CS course in their career. The following quotes from the student survey illustrate a desire to see more practical programming experience:

> *Some practical applications of basic programs.*

> *Practical application for future courses (non-CS).*

> *I believe that all the different departments should develop their own CS classes that are designed to teach the necessary programming required for their major.*

> *If the resources exist, it would be nice to have separate sections for the different majors so that students can apply their CS knowledge to their area of study.*

Although the definition of practical experience may vary from student to student (mostly depending on major), we believe programming examples which relate to non-majors' interests will increase their enthusiasm and will likely lead to better performance in the course.

Ultimately, non-majors should not have to take CS 1005 or CS 1006, as the objective of those courses is teaching computer science to students who have prior background in programming, rather than teaching programming to students with no prior programming experience.

2.2.3 Problems with CS 1001

The students who have taken it feel it's pretty useless.

–a WPI professor, regarding CS 1001

As previously mentioned, there exists a programming course designed specifically for non-majors called Introduction to Computers (CS 1001). The objective of CS 1001 is to "introduce computer systems to students who may need to write or use computer programs in their undergraduate engineering, science, or management courses." [18] From the course description, this course appears to be designed for the needs of non-majors.

However, we believe this course fails to meet the needs of most non-majors. According to the 2000–2002 WPI course evaluations, 203 non-majors took CS 1005 and CS 1006, versus only 13 who took CS 1001. We believe this is because the course is offered infrequently, is not well publicized, uses a language students perceive as unpractical, and leaves no option for further CS study.

A significant reason we believe non-majors do not take CS 1001 is because it is offered much less frequently than other introductory CS courses. CS 1005 and CS

1006 are each offered once per semester, while CS 1001 is only offered once every two years. This makes it difficult for non-majors to schedule the course. Additionally, many students do not even know the course exists, as seen from quotes received from our student survey:

> *I haven't found any classes meant for non-CS majors. One or two courses teaching practical programming techniques with minimal theory would be very useful.*
>
> *There really aren't any because if you take a CS course you are mixed with CS majors.*
>
> *I didn't know they [CS courses for non-majors] existed specifically.*
>
> *I do not know of the non-CS course, only of CS 1005.*

Obviously, students who haven't heard of the course will not take it. Another reason students may not know about the class is that advisors hesitate to recommend it. A WPI biology professor we interviewed said that the examples in CS 1001 are more aimed toward engineers, and that "the examples don't really speak to biology majors." The professor concluded that the course as a whole makes "no sense for biology majors."

One reason students feel the course is not useful is because it uses the FORTRAN programming language. FORTRAN used to be one of the primary languages used in the scientific and engineering communities, but in recent years its use has begun to decline. Although FORTRAN still is used, many students feel the language is out-of-date, as seen in these comments from our student surveys and from CS 1001 surveys:

From our student survey:

> *FORTRAN is basically a dead language.*

From a CS 1001 post-survey:

> *I feel a more widely used language might have been better such as C.*
>
> *FORTRAN is a useless language; it currently has little practical applications.*
>
> *FORTRAN will not help me in my field, it is obscure.*

Whether or not FORTRAN is a useful language for students to learn, the perception that it is not useful is discouraging students from taking CS 1001.

Finally, students who take CS 1001 and wish to continue studying programming are encouraged to take CS 1005 or CS 1006 next:

> *Students who develop interest in computer science after taking CS 1001 are urged to consider taking CS 1005 or CS 1006, followed by CS 2005.* [18]

A student who takes this path will already be one course behind, as the CS minor and CS concentration do not allow more than one 1000-level course to be counted. This will discourage students who are at all interested in a CS minor or concentration from taking this course. The current situation is a dead-end into a course described as requiring no prior programming background. Although there are difficulties in creating a valid course path from CS 1001 to more advanced courses, the lack of such a path discourages non-majors from taking the course.

2.3 Conclusion

We believe the following quote from our student survey summarizes the current situation for non-majors:

> *I think that there is no option for WPI non CS majors. FORTRAN is basically a dead language, and the C++ class is a CS class that CS majors have to take. So the non CS majors might need the class to go slower or less in depth and the basic concepts enforced more, while the CS majors may get bored.*

In order to create a solution to these problems, we have created a list of the five primary criteria we believe must be satisfied for a CS course to meet the needs of non-majors. These criteria are:

1. The courses should pertain to students' majors
2. The courses should not be intimidating or overly difficult
3. The courses should fit into schedule or distribution requirements
4. The courses should be enjoyable
5. The courses should focus on applied programming

We believe that these criteria must be met before an introductory CS course can successfully meet the needs of non-majors.

Chapter 3

Other Efforts

3.1 Introduction

Before attempting to define our own solutions to the problems we identified, we wanted to evaluate other introductory CS courses. We begin by examining WPI's newly-proposed introductory CS curriculum. This proposal is aimed at completely revising the introductory CS curriculum for both majors and non-majors. We feel that this new curriculum, as currently proposed, will not meet the needs of many non-majors.

We also examine introductory CS programs at other schools, at both the high school and college levels. In particular, we look at the Yorktown High School introductory CS course, the Centre College CS-I course, and the Princeton "Computers in Our World" course.

Finally, since many schools use courses based on the Scheme programming language, including WPI's newly proposed introductory course, we evaluate Scheme-based courses in detail.

3.2 WPI Proposed Curriculum

3.2.1 Description

Less theory, more problem solving exercises.

–a WPI non-major

As previously discussed, the WPI computer science faculty has noticed problems with the introductory CS courses. They followed up their observations with a proposal for a new curriculum. The new curriculum is meant to address the problems for both majors and non-majors. The curriculum's goals include emphasizing program design, making courses approachable for students with no prior programming experience, and postponing teaching difficult concepts until later courses. [17]

The proposed introductory sequence consists of three courses:

CS 1101: Introduction to Program Design (Scheme)

CS 2102: Object-Oriented Design Concepts (Java)

CS 2303: Systems Programming Concepts (C and C++)

The sequence is designed for both majors and non-majors. We are specifically interested in the impact on non-majors. The proposal describes the expected impact on non-majors:

> *Students who want a 1-term introduction to computing and programming should take [Introduction to Program Design]. This course will teach them enough data structures and program design skills for a variety of core computing tasks (in particular, students will be able to write more sophisticated programs than they can write coming out of our current*

> *1005 as a terminal course). Ideally, a strong intro to CS requires two courses. We recommend one of two 2-course sequences: CS 1101/2 and CS 2102 (to end in Java), or CS 1001/2 and CS 2301 (to end in C). [17]*

The "Introduction to Program Design", or CS 1101, course description is as follows:

> *This course introduces principles of computation and programming with an emphasis on program design. Topics include design and implementation of programs that use a variety of data structures (such as records, lists, and trees), functions, conditionals, and recursion. Students will be expected to design, implement, and debug programs in a functional programming language.*
>
> *Intended audience: students desiring an introduction to programming and program design.*
>
> *Recommended background: none. Either CS 1101 or CS 1102 provide sufficient background for further courses in the CS department.* [17]

3.2.2 Problems with WPI-proposed curriculum

The computer science department created this introductory sequence with both majors and non-majors in mind. We believe this sequence will work very well for CS majors and minors, but not for non-majors.

The proposed introductory sequence is intended to meet the needs of both CS majors and non-majors, but it does not accommodate the needs of non-majors. The needs of CS majors and non-majors differ. The goals of teaching a future computer scientist are for the student to understand the concepts and theory behind programming and program design. These ideas provide a foundation for continuing study in the computer science field.

Non-majors take a computer science course to learn basic programming skills so they can apply programming techniques to automate, simulate, and calculate. These skills will give students an advantage in their field of study. The goals of a computer science major and a non-major are different. We believe that both cannot be successfully addressed by one course, and that treating non-majors the same as majors will result in unsatisfactory experiences for both.

We believe that separate courses for majors and non-majors would provide a positive environment and atmosphere for each. Having an integrated course can lead to an imbalance in the performance of the students (as was already demonstrated in CS 1005 and CS 1006) [17]. The WPI CS proposal believes this problem is addressed because the initial course is in Scheme. They believe that, since few students coming from high school have experience with Scheme, all students, major or non-major, will enter with equal background in Scheme, therefore there will not be an imbalance. We believe there will still be imbalance in an integrated course. From our student survey, a number of students want to see a very basic introductory CS course for non-majors, as illustrated in this quote:

> *Very, very basic introduction designed for people like me whose computer knowledge extends to typing up papers.*
>
> –WPI non-major

To accommodate the wishes of non-majors, an introductory course would have to start at a slow pace. If majors are subjected to such a slow start then they will likely become bored and quickly lose interest in the course, which is particularly harmful with WPI's seven-week terms. The converse is also true. Non-majors taking the course who are not familiar with computers or even familiar with a programming language will become lost in a faster paced course and give up, similar to what we see with CS 1005 and CS 1006 today. Furthermore, some non-majors we interviewed indicated that they were intimidated by being in the same class as CS-majors. Faculty members who participated in our faculty survey also mentioned that intimidation is a factor, reinforcing the students' comments:

> *I've also spoken with non-CS majors who would like to take a programming course, but not with CS majors. They feel that if they took a course with intense CS majors, they would be left behind and they wouldn't end up learning much. It seems to me that their needs may be very different.*
> –WPI Professor

We believe that having separate courses is the best option for those students who want an introduction to CS, as well as those students who wish to pursue a career in computer science.

Certain concepts are particularly useful for students who wish to use programming in the fields of science and engineering. Examples of such concepts include:

- Numerical methods
- Computer simulations and modeling
- Scientific functions and libraries
- Computer precision
- Reading and writing data sets
- Displaying data visually

It is doubtful that a combined intro CS course has the resources to address topics which apply more to non-majors than to majors.

WPI CS proposal research

The CS department's proposed CS curriculum includes the removal of CS 1001 and the creation of a combined introductory course for majors and non-majors. We believe that the CS department did not adequately research the problems which affect non-majors taking introductory CS courses. The CS department only directly contacted departments "whose curricula depend on CS courses." [17] We believe the CS proposal focuses on these departments to the exclusion of others. Although they "solicited feedback ...from departments heads," as far as we know, the CS department did little direct research as to what other departments are looking for in an introductory programming course.

The CS department proposal does not mention any attempt at student feedback. As computer science students, we were surprised to discover a new proposal to restructure the introductory computer science courses. As far as we know, there was no attempt to contact the student body to get feedback on the proposed plan. We feel that student input is an important aspect of research when deciding to significantly restructure courses.

We feel the lack of local research done by the CS department results in flaws in their proposal and neglects the needs of non-majors. We have done a significant amount of research to get the feedback of both students, faculty, and industry professionals. Because our recommendations are based directly on feedback from non-major students and faculty, we believe our proposal is a better solution for the needs of non-majors.

Scheme

Another disadvantage we see with the proposed curriculum is that it is taught in Scheme, a language which we do not feel is appropriate for WPI's non-majors. This problem is discussed in detail later on in this chapter (see section 3.4).

3.3 Other Schools' Efforts.

3.3.1 Breadth-first approach to CS-I at Centre College

One of the programs we examined was the introductory course in computer science (CS-I) at Centre College, in Kentucky. [1] The Centre College course takes a breadth-first approach, covering a wide range of topics from all areas of computer science. The course was designed to meet the following objectives:

1. Students will learn an object-oriented language, including the use of classes and a graphics package.

2. They will better understand how computers work (including finite state machines, logic gates, nature of algorithms, and limits of computation).

3. They will demonstrate an understanding of the operation of the Internet and the World Wide Web.

4. They will be able to use HTML to construct a web page containing a form and write a CGI script to process the information that is submitted with that form.

5. Students will think critically about ethical questions and societal concerns that arise in the context of information technology.

[1] `http://www.centre.edu/`

6. They will explore the area of artificial intelligence and robotics
7. They will write simple database queries using SQL and learn the basic vocabulary of relational databases. [14]

Additionally, the class is a combination of majors and non-majors (due to lack of resources), and the course uses the Python programming language.

We feel that the wide range of topics covered in the CS-I course simply could not fit into one of WPI's seven-week terms. Furthermore, we feel that the stress placed on topics such as the societal impact of computers and the study of databases is of less interest to students studying science and engineering, and that those topics would detract from learning more applicable concepts.

However, the Centre College approach meets several aspects of our criteria. They reported success with teaching a programming-intensive course where "at least half of each such course consisted of students whose primary interests ranged from mathematics and the sciences to drama and other fine arts." [14] The primary reason listed for this success was the switching of programming languages from C++ to Python:

> *A course that emphasized object-oriented design and programming in the language C++ did not seem like the most appropriate experience for a large portion of our audience. Many simply could not manage the difficulty of programming in such a complex language.*

Another reason why the course was successful was because of the variety of topics covered:

> *Students liked the variety of topics – especially the graphics and material related to the Internet.*

Although a course at WPI would not be able to cover the breadth of the Centre College course, we believe that both graphics and Internet programming would appeal to the non-majors at WPI. Additionally, Centre College noted some success with programming robots, which we feel may be appropriate considering the number of engineers at WPI.

3.3.2 Princeton liberal-arts computers course

The "Computers in Our World" course at Princeton aims to "demystify computing for a classroom full of liberal arts undergraduates." This course gives students experience in creating web pages and writing a few simple programs, but focuses on technology's impact and practical computer knowledge.

While the course's goal, "to make it possible for [students] to think intelligently about this technology for themselves", is certainly one of the goals of our course, we do not believe that such an overview of computer science is appropriate for WPI students. WPI has a tradition of being a practical, hands-on institution. [2] Furthermore, the professors we interviewed stressed the fact that they wanted practical programming ability and not a "CS for poets" course.

3.3.3 Yorktown High School

Another school we studied in detail is Yorktown High School in Arlington, VA. The Yorktown High School introductory programming course, taught by Jeffrey Elkner [3], uses the Python programming language to teach programming to high school students. This program is discussed in more detail in section 4.4.2.

[2] See http://www.wpi.edu/Academics/Library/Archives/Tower/ for details.

[3] http://www.elkner.net

3.4 Scheme-based Programs

There are over fifty colleges and universities in the U.S. that teach introductory courses using the Scheme programming language. [13] As previously mentioned, WPI's newly proposed introductory course will be offered in Scheme.

3.4.1 Scheme introduction

Scheme is a programming language based on an earlier language called Lisp. It was designed to have clear and simple semantics and few different ways forming expressions. It is often used in computer science curricula and programming language research.

3.4.2 Strengths of Scheme-based approaches

The TeachScheme! project is a Scheme-based introductory curriculum which is growing in popularity. TeachScheme! uses a novel approach to problem solving called design recipes:

> *We created the design recipes by identifying categories of problems. The identification of a problem category is based on the classes of data that are used to represent the relevant information. Starting from the structure of this class description students derive the programs with a checklist.*
>
> 1. *the description of the class of problem data;*
> 2. *the informal specification of a program's behavior;*
> 3. *the illustration of the behavior with examples;*
> 4. *the development of a program template or layout;*

5. *the transformation of the template into a complete definition; and*

6. *the discovery of errors through testing.* [7]

We believe this is an excellent approach to problem solving, and hope to incorporate it into our proposed course.

3.4.3 Problems with Scheme-based approaches

Although we believe Scheme has many good qualities, we do not believe any language is appropriate in all cases. In particular, we do not believe that Scheme will adequately meet the needs of non-majors at WPI. This is due to its unfamiliar syntax, lack of use in industry, focus as an educational and research language, and lack of traditional looping constructs.

Scheme is infamous for its syntax; unlike more common languages (such as Java, C++, BASIC, and MATLAB), Scheme uses prefix notation with surrounding parentheses. For example, the calculation

```
5 * 2 + 1
```

would become

```
(+ (* 5 2) 1)
```

We believe that this syntax is generally more difficult to understand for a beginning programmer, which could lead to an initial intimidation regarding the language. For example, asked to compare programs to calculate the quadratic equation in both Scheme and a language with C-like syntax (Python), 67% of non-majors preferred Python, versus 5% who preferred Scheme. [4] Although most students will be able

[4] From student survey results (Appendix A.1.2)

to learn the syntax of Scheme, we believe that the syntax will intimidate many students.

Furthermore, if a non-major successfully learns the Scheme syntax, he will need to learn another syntax if he begins to program tasks in his major. In a survey given to WPI faculty, many professors indicated that "using a language with syntax similar to C/Maple/MATLAB" is important or very important for their students.

One of the features advertised as an advantage of Scheme over Lisp is that it "[relies] entirely on procedure calls to express iteration ..." [12] In simpler terms, this means that Scheme can iterate only by using recursion; it does not have loops such as *for* or *while*.

Although recursion is a useful technique, the vast majority of languages primarily use looping constructs for iteration. Additionally, professors we interviewed and surveyed indicated that understanding common looping syntax was important material they would assume from an introductory programming course. For this reason, we believe Scheme is a poor choice to teach non-majors.

Scheme is not a language that non-majors are likely to see outside of their introductory course. Although Scheme is used occasionally (usually as an extension language), "there are few known uses of Scheme in 'real-world' systems." [12] Even if they do encounter the language, it is likely to be a different version, as there are over fifty different implementations. [12]

Finally, Scheme's primary use is for "computer science curricula and programming language research." [12] This is because Scheme is a very abstract language, and well-suited to CS theory. Since the language's focus is on theory, rather than practical programming, curricula based on Scheme tend to focus on theory as well. Many introductory Scheme programs use the freely available *How to Design Programs* text. To quote from this book,

> *Still, the book [is] not about programming in Scheme. We only use a small number of Scheme constructs in this book. Specifically, we use six constructs (function definition and application, conditional expressions, structure definition, local definitions, and assignments) plus a dozen or so basic functions. This tiny subset of the language is all that is needed to teach the principles of computing and programming.* Someone who wishes to use Scheme as a tool will need to read additional material. *[emphasis ours]* [7]

This quote illustrates that many Scheme-based programs focus on computer science principles and concepts, rather than applied programming. For example, *How to Design Programs* avoids covering file input and output, a programming concept essential for non-majors who will write application programs in their field.

For these reasons, we believe that Scheme is an inappropriate language for an introductory CS course for non-majors. It does not fulfill the level of applied programming which most WPI non-majors would require.

3.5 Summary of Other Efforts

None of the programs we examined fulfill our requirements for a CS course for non-majors. However, certain aspects of these programs are very applicable to our goals, and we have incorporated them into our course proposal. These include using a subset of a breadth-first approach taken by Centre College, using design recipes, and using the Python programming language.

Chapter 4

Recommendations

4.1 Introduction

In order to fulfill the criteria we have established, we have created a set of recommendations for the creation of a successful CS course for non-majors. Our ideas are based on results from our student and faculty surveys and interviews, and from our personal experiences.

Primary recommendations include major-based projects, use of the Python programming language, daily assignments, and different course listings to ease scheduling conflicts for non-majors. We believe these solutions help meet the programming needs of non-majors.

4.2 Major-based Projects

If the resources exist, it would be nice to have separate sections for the different majors so that students can apply their CS knowledge to their area of study.

–WPI non-major

One of our main ideas for a new CS curriculum for non-majors is the concept of *major-based projects*. Instead of assigning all students the same programming project, as is traditionally done in CS courses, projects would be assigned to students based on their major field of study. That is, a physics student might be assigned a projectile-motion programming project at the same time as a biology major is assigned a population-growth problem. We borrowed the idea for this approach from Professor John Goulet's Linear Algebra course at WPI.

4.2.1 John Goulet's Linear Algebra course

Professor Goulet has been using major-based projects successfully in his Linear Algebra course for the past four years. He has structured his course into two components,"a core component covering traditional mathematics. . . , and a project component organized according to major." [9] Students are divided into groups and are assigned projects based upon their major.

Goulet's goal was to "do as much as possible to relate linear algebra to each student's chosen major." [9] He says that the new course receives "a lot of effort and a lot of enthusiasm out of people." Students also like the project; comments from course evaluations include:

> *The projects were major-oriented, so it made it a little more realistic for everyone.*
>
> *Projects relate this class to ECE well.*
>
> *The material is very applicable to other classes.*
>
> *The projects were good tools to show how [linear algebra] can be used in other subjects.*

Bridges learning of linear algebra with electrical engineering by use of Fourier Series.

Furthermore, Goulet has seen increased exam scores since he has implemented the major-based project system [9], showing that students are learning more linear algebra as well. Another benefit of Goulet's project system is that "the project work [is] done entirely outside of class." [9] Goulet states that this gets students to spend more time on linear algebra during the week, while keeping them interested–a result he feels would be impossible with homework alone.

We believe that implementing a similar major-based-project structure for an introductory CS course for non-majors will see similar success. There are many similarities between CS courses for non-majors and linear algebra courses; both are taken primarily by freshman, both cover useful tools which apply to most scientific and engineering disciplines, and both are taken by a variety of different majors. Most computer science courses are project-based already, so the changes to the structure of the course are less than that of linear algebra.

4.2.2 Proposed format

In creating these projects, we have attempted to create a standard format to present the problem to the students. Our method for constructing these problems was influenced by the TeachScheme! design recipes concept. Each project will consist of the following sections:

1. A list of the concepts and techniques used

2. A problem statement

3. A defined programming project

4. (Optional) Required background

5. A set of open-ended questions

6. Sources of additional information

For full project examples, see Appendix C.

List of concepts and techniques

The first section of each project will be a list of the primary CS, programming, and scientific techniques required to solve the project. An example for a mathematics-based project might include

- Integrals
- Numerical Methods
- Functions/Subroutines
- Flow control
- Random numbers

Problem statement

The second section is a statement of a problem a student might encounter in his major. The problem statement should not mention the actual programming assignment; it should only describe a problem the student might encounter in his major. However, the problem would ideally either require, or be greatly simplified by, the use of programming.

Defined programming project

The next section of each project would help in the transition from the problem statement to an actual programming project specification. For the first project, a student would be given specific details on how to convert the problem into a computer program. However, the eventual goal is for the student himself to be able to create a program outline directly from the problem description. This section would still have hints, and list some final goals, but subsequent projects would not be as explicitly defined as the first project.

(Optional) Required background

Since the students working on these projects will come from different class years and will have had varying exposure to their actual major material, projects will mostly be chosen from high-school or freshman-level material. However, other material may be used, as long as the topic is simple enough and makes an appropriate programming assignment. In such a case, the required background will be given in the fourth section. Additionally, any programming techniques which are required but not covered in class will be explained (for example, additional modules).

Set of open-ended questions

In addition to the programming project, the student will be given a set of questions to answer. These questions will cover concepts regarding both the original science or engineering problem, and the programming project. The purpose of these questions is for the student to think about the interaction between the problem statement and the programming project and the limits of programming. Some examples include asking how a simulation might be improved, or asking for an example of a problem that might be difficult to solve using numerical analysis.

Sources of additional information

The last section of each project deals with providing additional information. This would include additional help resources (such as web pages), or pointers for more information about the major and computer science topics covered in the project.

4.2.3 Designing projects

The major-based projects we propose will require a number of project ideas per major. Initially designing these projects would therefore take a substantial amount of effort on the part of the professor teaching the course. We hope to reduce the amount of work required by approaching the various departments on campus, and asking for their support in providing problems we might develop into major-based projects. Feedback from faculty surveys indicates that most departments will be willing to provide such support.

There are also currently many programming books published which are for "scientists and engineers." [1] These books contain programming projects which can be converted in language and depth to fit the proposed project format, with a minimum of difficulty. Once a project list is compiled, it should not require a great deal of effort to keep it up-to-date.

One idea for helping with organizing the different problems is to create a web-based problem repository. This web site would allow professors from various departments to share project ideas online. Such a project repository could in theory be used by many different departments, for both intra-departmental projects, and collaboration between departments. Although the design of such a project repository is outside the scope of this IQP, it is an interesting idea for future research.

[1] Such as *FORTRAN 90 for Engineers and Scientists*; see the bibliography for more examples.

4.2.4 Possible pitfalls

In addition to the difficulty of creating project ideas, considered above, there are other possible pitfalls we must consider. In order for the projects to be fair, the projects assigned to different majors must all be at about the same level, and must utilize most of the same programming concepts. Projects for a specific major must deal with concepts that are familiar to all members of that major. We must also deal with the problem of students who have majors for which there is no project defined, or who are undeclared.

There may be some difficulty in creating projects which are at the same level for each major. We hope to alleviate this problem by focusing on a main set of programming concepts which will be covered in every major for each project. For example, the second project for every major might cover arrays, random numbers, and functions. Although this approach could add some programming requirements which are outside the scope of the original problem, we believe that effect will be minimized.

Another possible problem, which has already been mentioned, is ensuring that the level of major-specific material is appropriate for all students of that major. The problem, therefore, should not assume knowledge beyond the high school or freshman level. We hope to design the projects so that they provide any material that the student may not have learned already; at least enough for the student to be able to fully complete the project, and have a reasonable idea of what the project is about.

One feature which we believe will be very important is the extensive use of extra credit in the projects. It is our goal to create a course which accommodates students at different skill levels, both in terms of major-ability and programming ability. In such a course, the base level must accommodate students who are intimidated by programming. However, it is important for students who find that they enjoy programming to have an opportunity to do work at a more advanced level. Otherwise, students might become bored with the course, and subsequently the course would suffer.

WPI currently offers more than thirty different majors, so the task of creating a separate assignment for every one is probably not feasible. In these cases, there will usually be a project in a topic which is related to the major in question; for example, some engineering disciplines may be combined at first. As more students from a particular major begin taking the course, creating a separate set of projects for that major should become a priority.

Many WPI students have dual majors or are undeclared. In the former case, they could pick a project from one of their two majors. In the latter case, they could choose a project in a major that they are considering. Another solution is to allow students to choose any project they wish, independent of their major. In addition to allowing for the cases above, allowing the students to choose would be a check on the fairness of the projects. If several students are avoiding a specific project, or are choosing a specific project, then that project may be too difficult or too easy.

Allowing students to choose their own project, however, conflicts with fundamental ideas of major-based projects. Additionally, there could be many problems if students continually switch their project-major. For example, if the projects are group-based or if the projects build upon one another, then it would be difficult to switch majors midway. Also, since there will always be some variation between the projects, a student might attempt to always take the easiest project. For these reasons, we recommend that students should be allowed to select their project, yet should be discouraged from choosing one outside their major.

4.2.5 Expected benefits

We believe that implementing a major-based project system would greatly increase student interest in programming and computer science; that it would accelerate the learning process; and that it would allow students to become comfortable and confident in their programming ability. We fully expect results similar to those Professor Goulet saw with using major-based projects in his Linear Algebra course.

4.2.6 Evaluating the major-based project idea

One of our goals in this IQP was to test out the idea of major-based projects in the CS area. Although Goulet's Linear Algebra course has shown that the idea has merit, we hoped to test the idea in a computer-science setting. In order to accomplish this goal, we needed an introductory computer-science situation in which we could replace regular projects with major-based ones.

Professor Joe Wong of WPI was willing to implement some of these changes in his CS 1001 (Introduction to Programming in FORTRAN) course. The CS 1001 course usually has an assignment every week, which consists of three programming problems. Wong first changed this format slightly, giving two required problems, and allowing students to choose a third problem from a set of major-based problems. We then worked with Professor Wong to design a set of two-part major-based projects which represent the most of majors in his class.

Actual results

We evaluated the major-based project idea with the help of Professor Wong. Each project we gave to Professor Wong contained a set of questions to get feedback from the students. The questions we asked the students were:

- Why did you choose this project?

- How did this project compare to other assignments you have had so far in CS 1001?

- Has this project stimulated your interest in programming?

We only received five responses from the students. Two were positive, two were neutral and one was negative. All students chose a project because it related to their major. All but one student noted that the project was more difficult then previous projects. The project difficulty caused a few students to react negatively to the project. As one student stated:

> *It was much more interesting [than] other assignments, but a lot more time consuming. Honestly, due to the hours of debugging it caused me to [endure], it demotivated my interest in computer programming.*

Another student describes the project

> *This two-part project was larger and generally more challenging than the other projects I have completed for this class ... Working hard and completing this program gave me a sense of satisfaction since I was interested in the topic I was dealing with.*

Arriving at a definitive conclusion with such a small data set is difficult. The negative and neutral responses were mainly due to the difficulty of the project. The positive results were encouraging. The students were interested in how computer science relates to their major. As one student states:

> *This project has shown me just how useful programming can be, even to a biology major.*

4.3 Variations on Projects

There are many possibilities which could be used to enhance major-based projects. Professor Goulet's Linear Algebra projects are both group-based and cumulative; Goulet feels these are major points of his program. Additionally, we wish to discuss the idea of creating interdisciplinary projects.

4.3.1 Group projects

Although we have not specified whether major-based projects will be done in groups, we feel there are many benefits of group work. Since WPI requires many projects, usually done in groups, prior group experience is directly beneficial. Group projects encourage team work and cooperation, which are useful job skills. Finally, when students work in a group, they don't have the feeling that they are going through a course alone.

4.3.2 Cumulative projects

Cumulative projects are projects which build upon one another. This allows the student to create progressively more complex programs, and allows the student to better apply one assignment to the next. The final project would represent a substantial accomplishment, which we believe would be more satisfying to the student than small, independent projects.

4.3.3 Interdisciplinary projects

Our final idea is for interdisciplinary projects. The idea would be to create a complex problem which involves several different majors. The students would then need to work together on their respective parts to create a complete solution. We believe this idea, if implemented, would provide an excellent teamwork experience.

4.4 Python

4.4.1 Why a specific language?

Before explaining why we recommend Python, it is reasonable to ask why we recommend a specific language at all. Professors implementing our proposal may prefer to use another language, and there are certainly other languages which could be appropriate for our course. In defining some aspects of this course, we have attempted to keep the course as language-neutral as possible. However, we decided to recommend a specific language in order to establish a reference for the course and to take advantage of some features specific to the language. Additionally, we believe that many commonly used languages are not optimal for teaching non-majors.

In many ways Python serves as a reference language for this project. We believe that our final course recommendations could be adapted to another language, with the degree of difficulty being dependent on the language chosen. Choosing a specific language allows us to give concrete examples of code. This makes the examples more useful, and provides background for an actual implementation of the course. Furthermore, we chose a language with a very simple syntax, so that translation into another language would require minimal effort. Using a specific language also gives a model to refer to when giving a sample course outline; the order of topics covered is related to the language chosen.

Choosing a specific language allows us to take advantage of the features of the language. Languages each have their strengths and weaknesses; in our report, choosing a specific language allows us to capitalize on its strengths. For example, we plan to incorporate specific visualization and mathematics libraries, and to take advantage of certain built-in data structures.

Although we believe our ideas for this course can be adapted for other languages, we believe that the choice of language is extremely important for a course for non-majors. We have identified the following points as being crucial for a language to be useful for non-majors:

- The language should be accessible
  - Free for students
  - Cross-platform
- The language should be easy to use
  - Simple syntax
  - Easy to run
  - Integrated development environment
- The language should have good library support
  - Graphical libraries
  - Advanced mathematical libraries
- The language should have syntax similar to languages used in industry
  - Similar to C, Java, or MATLAB

We believe that a language should be free and cross-platform to allow the students to work from their own computers, and to allow students to continue using the language after taking the class. This is the primary reason we avoided languages like Maple and MATLAB. However, if these languages were provided free-of-charge to the students (for example, if the school possesses a site license), these languages could be considered.

The language chosen should also be easy to use. We define ease-of-use to include a simple syntax, a simple running procedure (i.e. no extra compilation steps), and possibly an integrated development environment (IDE). Simple syntax is important for any beginning programmer; the student should not be burdened with unnecessary overhead in order to write simple programs. Likewise, a student should be able to run his program with minimal effort. For this reason, we believe a language which does not require separate compilation steps is the best choice for an introductory course. Finally, we believe that the use of an IDE can greatly reduce the frustration of learning programming. An IDE is an application that helps the programmer develop his programs; it helps with tasks such as indentation, and allows the program to be run and tested without starting another program. Without an IDE, students must either use a text editor which is not suited for writing code, or use a complex general-purpose editor such as emacs. The IDE, like the language, should be free for the students.

The last item we feel is important for a language to be appropriate for an introductory course for non-majors is a rich set of standard libraries. Libraries are prepackaged tools, which allow the student to extend a program without writing all the code himself. Examples include advanced mathematics, visualization, and data structure packages.

We believe that choosing a particular language enhances and completes our recommendations. Although our recommendations could be applied using a different language which meets the criteria we have given, we believe the language we have chosen is the best match to the goals of this course. A comparison of how different languages meet our criteria is given in Table 4.1.

Language	C++	MatLab	Python	Scheme
Free	Yes	No	Yes	Yes
Simple syntax	No	No	Yes	Yes
No separate compilation steps	No	Yes	Yes	Yes
IDE	Yes	Yes	Yes	Yes
General purpose language	Yes	No	Yes	Yes
Visualization libraries	Yes	No	Yes	No
C-style syntax	Yes	Yes	Yes	No
Used in industry	Yes	Yes	Yes	No
For, while loops	Yes	Yes	Yes	No

Table 4.1: Comparison of Programming Languages

4.4.2 Introduction to Python

Python greatly simplifies programming examples and makes important programming ideas easier to teach.

–Jeffrey Elkner, Yorktown High School teacher

Instructors noticed that the level of enthusiasm was up and the level of frustration was down.

–Christine Shannon, Centre College

As previously mentioned, Python has proved to a very effective teaching tool. Many schools have instantiated introductory programming courses using Python. [2] The most documented case study is the introductory programming course at Yorktown High School in Arlington, VA. Jeffrey Elkner, the creator of the programming courses in Python at Yorktown, has written several reports about his experiences. According to Mr. Elkner, he chose Python for his course because, "Python greatly simplifies programming examples and makes important programming ideas easier to teach." [4]

[2] For a list, see `http://www.ibiblio.org/obp/pyBiblio/schools.php`

Since the introduction of Python in Elkner's classroom, he has noticed increased enthusiasm among his students and a subsequent increase in enrollment. The interest in learning programming has increased so much that the maximum size of the class was increased, and students are still being turned away. [6] Elkner explains the impact on his class by saying, "Increased enrollment naturally follows increased student interest, which in turn results at least in part from greater student success as made possible by the use of Python." [6]. Students who take Elkner's course don't stop programming when the semester is over. Many students continue their projects outside of the classroom. Some examples of projects that Elkner's students have worked on are:

pyKarel a robot simulator. [3]

Zuite a database system for handing in homeworks and keeping up with class, being used by four courses at Yorktown High School. [4]

SpellQuest a computer-based learning program for studying spelling words. [5]

pyJotto a Python implementation of the word game Jotto. [6]

In an interview [7] with Elkner, we asked him how the students' projects have changed since he began teaching Python.

> *With Python programming is faster to learn and easier to be productive. Students are now capable of doing things they could not have done before. Three students this year are working on pyJotto, and program*

[3] `http://pykarel.sourceforge.net/`

[4] `http://openclassroom.sourceforge.net/`

[5] `http://spellquest.sourceforge.net/`

[6] `http://pyjotto.sourceforge.net/`

[7] for full transcript of the interview see Appendix B

> *called SpellQuest. In each of these projects, I have been able to hook up interested students with professional programmers, who mentor them.*

As previously mentioned, Centre College in Danville, Kentucky recently changed its introductory CS course language from C++ to Python. Both students and faculty at Centre College have reacted positively to using Python. The faculty have attributed their success with Python to its simple syntax and ease of learning:

> *[Python] rates high on the expressiveness index. Even beginning programmers can complete interesting and significant projects very quickly. The syntax is very simple and the structure is uncomplicated. This makes code both easy to read and write. Students pick it up very easily.* [14]

One of our criteria for evaluating a successful CS course for non-majors is for the students to feel comfortable with programming, rather than intimidated by it. After switching to a Python-based course, Centre College noted improvements in this area:

> *Student evaluations from the first three times the course was offered were very positive. ...Instructors noticed that the level of enthusiasm was up and the level of frustration was down.* [14]

We believe the Yorktown High School and Centre College programs have proven that Python can be used successfully in an introductory programming course. Additionally, as both programs contain a high percentage of non-majors, Python has proven to be an effective language for non-majors specifically.

4.4.3 Visual Python

[Visualizations] would be really good for MQPs–a lot of people could use that.

–a WPI chemical engineering professor

Visual Python, or VPython, is a visualization tool for Python. [16] It eases the process of programming graphics. Graphics can be used for modeling environments or plotting data. VPython has been used in an introductory physics course to do modeling [16]. Visualizations provide a medium for students to interact with a model and immediately observe the results on a computer screen.

4.5 Daily Assignments

In an introductory course, constant exposure to the subject matter is critical. This is especially true of WPI's seven-week terms. If students are well versed in the basics of a subject they will be able to better understand advanced topics, and they will be able to continue to learn on their own. If they are not well-versed in the basics, their confidence can suffer, resulting in frustration and poor performance. We believe this problem affects many non-majors in the current CS introductory courses.

We want to address this problem by assigning regular homework assignments. The assignments will be given daily, or on alternating days, at the beginning of the course, then be reduced in frequency to one or two per week, depending on how the professor wants to conduct the class. The assignments are meant to be 'easy,' since we want to get the students familiar with the programming environment and build their confidence. Therefore each assignment should take no more than an hour of the students' time.

One of the main problems of introductory CS courses at WPI is that most assignments are projects. This means that the student only has a few, large assignments. This format allows the student to procrastinate until the last minute, when he must quickly finish a project intended as a week's worth of work. We believe that this approach does not give the student a good exposure to the language.

Daily or frequent assignments, on the other hand, allow the student to incrementally learn and practice programming. We believe that this will allow students to become familiar with the programming environment. Additionally, the constant exposure will familiarize them with the language as well as with common errors.

As CS majors, we have also noticed that project-only courses tend to create a large time lag in terms of getting feedback regarding a project. For a beginning student, this is unacceptable; if a student is doing something wrong, or misunderstands a concept, it is essential that he receives feedback on his mistake as early as possible. Frequent assignments address this problem by giving the student constant feedback, with a faster response time.

4.6 Satisfying Distribution Requirements

One reason non-majors do not take CS is because a CS course does not fit well into their distribution requirements. Most majors do not have a CS requirement, and some do not have an appropriate space for a CS course. This is most evident in the chemical engineering (CM) major. Out of the twelve chemical engineers we surveyed, nine listed "no room in schedule" or "not required" as the primary reason for their not taking a CS course, rather than "no interest." In fact, some of the students indicated that they would like to take a CS course if they could fit it into their schedules. A chemical engineering professor at WPI pointed out in an interview that the chemical engineering major has little room for CS electives, but does have an engineering sciences requirement. She said that offering the course as an engineering science (ES) credit would be more appealing to chemical engineering majors.

Cross-listing an introductory programming course for non-majors as both CS and ES would allow more students to count it toward their degree requirements, and would therefore increase enrollment in the course. Additionally, we believe that such a course would be appropriate as an ES course, since programming is a useful engineering tool, and because many ES courses require programming background. It may even be possible to allow ES credit only if a student completes engineering-based projects.

Another possibility for increasing enrollment is to encourage various departments to consider requiring a computer science course. Professor Camesano stated that "many other schools require programming in chemical engineering." Some majors already require a CS course, such as mathematics, MIS, and ECE. Since computers are such an integral part of modern life, it may be reasonable to consider requiring students to have some programming experience. Although such an effort is outside the scope of this project, having an appropriate programming course for non-majors may be a prerequisite of such a plan.

4.7 Summary

It is our belief that these recommendations each solve several of the problems we have noted in the current and proposed WPI CS course curricula, with respect to non-majors. Taken together, these recommendations fulfill our established criteria for a successful CS course for non-majors.

Chapter 5

Proposed Course

5.1 Course Description

CS 100X.

INTRO TO PROGRAMMING FOR SCIENTISTS AND ENGINEERS.

Cat. I

This course establishes a practical programming background for students who want to learn to write computer programs for their undergraduate engineering, science, or management courses. Topics include logical problem-solving and algorithm development, program design, debugging, language syntax, and error interpretation. Specific topics include control structures, functions, arrays, and simple I/O. Students will be expected to implement a variety of programs, both on their own and in groups, using the Python programming language. Group projects will be based on students' majors.

Intended audience: non-computer science majors desiring a practical introduction to programming. This course is not sufficient background for most advanced computer science or computer engineering courses. Such background may be obtained by taking CS 1005 or CS 1006 followed by CS 2005.

Recommended background: none.

5.2 Course Syllabus

Required Text: *How to Think Like a Computer Scientist*, by Downey, Elkner, and Meyers.

Class	Topics	Reading
1	Introduction, demos, Why is programming important to you	1
2	Variables, syntax	2
3	Variable types, assignment, simple math	2
4	Simple functions, more math	3
5-6	Booleans, conditionals	4
7-8	Fruitful functions, design recipes	5
9	Complex numbers, random numbers	handout
10	Loops (while)	6
11-13	Lists (arrays), loops (for)	8
14	Strings	7
15	Dictionaries	10
16-17	Graphing	handout
18	File I/O	11
19	Pattern matching	13, Appendix D
20-28	Advanced topics	

Table 5.1: Course Syllabus

Chapter 6

Conclusion

6.1 WPI Course Possibility

Our project was started with the intention of creating a course for non-majors to be offered by WPI's computer science department. Our research covers the feedback from the undergraduate community as well as feedback from the WPI faculty and feedback from professionals. Our recommendations reflect our research and, as a whole, encompass our course proposal.

We believe our recommendations will work extremely well for non-majors who wish to learn how to program and should be adopted by the WPI computer science department. We plan to present proposal to the computer science undergraduate committee in the fall of 2003.

6.2 Incorporation into Other WPI Courses

We believe that the recommendations we present would create an excellent introductory course for non-majors. However, many of our recommendations could be incorporated into the existing and newly proposed CS courses at WPI. We believe that, if successfully adapted, these recommendations would result in an improved experience for non-majors taking these classes. This would in turn lead to an increase the number of non-majors taking the courses.

6.3 Future Research

There are many areas of interest related to this project which we were not able to investigate in detail. The first of these is the creation of a web-based project database. This database would allow professors from various departments to submit assignments and projects. These projects could be designed for courses in the professors' department, or for interdisciplinary courses such as our proposed course. We believe that such a database would be very useful for both our proposed course and existing courses.

Additional research could also be performed regarding how non-majors respond to different languages. One possibility would be to have groups of students learn the same problems in different languages. Evaluations could be made to determine how each language impacts the students.

There are several opportunities for potential research topics in the area of major-based projects. In particular, the major-based project concept could be expanded to include group, cumulative, and interdisciplinary projects. In addition, the possibility of group-based projects in fields other than linear algebra and computer science could be examined.

Another area of interest, which we were not able to pursue in detail, is the concept of creating a support framework for non-CS professors who want to include programming assignments in their own courses. The framework would provide assistance in designing problems, helping students, and grading the projects. Support could be offered by teaching assistants or senior assistants, although interdepartmental funding issues would need to be considered.

6.4 Personal Reflections

We are particularly happy that we were able to design our own IQP. Although we were helped tremendously by our advisors, we chose our project idea, defined the project objectives, and were the main directors of the project. We would like to express thanks to our advisors for giving us the freedom to design a project.

Our primary goals in designing this project were to create a project that would be challenging, that we would find academically interesting, and that would actually be used. We believe that this project achieves our first two goals. We sincerely hope that this project will also be useful, ideally as the basis for a CS course for non-majors. We would also be pleased if some of our recommendations made their way into other CS courses, or if our IQP work were continued by other students.

Our initial project goal was to design a course for non-majors to replace CS 1001. However, after beginning this project, we discovered that there was a separate proposal put forth by the CS department which dealt with creating a new introductory curriculum for both CS majors and non-majors. As CS majors, this news surprised us, since the proposed curriculum was not widely publicized during its design phase.

The most frustrating part of this project for us was not knowing whether or not our course proposal would ever be considered by the WPI CS department. This caused our project definition to keep changing, and is one of the primary reasons our final project report consists of recommendations rather than a specific course proposal. Another problem we encountered was difficulty in communicating with faculty. Although all the faculty we dealt with were extremely pleasant to work with, it was difficult to get in touch with many of them. This caused problems earlier in the project's time line, as we tended to wait for professors to respond to email instead of going to talk to them directly. However, as the year progressed, we became better at tracking down faculty.

Although we did not create a complete course for non-majors, we investigated many ideas regarding how computer science can be better integrated with the specific needs of WPI non-majors. We believe our final set of recommendations contains innovative ideas, and that a course based on such ideas would achieve our original goal of a successful course for non-majors.

In the end, we feel our project turned out very well, and we are happy with the result.

Appendix A

Survey Results

A.1 Student Survey

There are problems with the current curriculum. That has been established from our results as well as the CS department's own observations throughout the years [17]. It was important for our IQP to get feedback directly from the students to evaluate the problem with the current situation.

We issued a survey to students to evaluate the current situation for the computer science introductory sequence. Specifically we were looking for data on the courses that non-majors take, which include CS 1001, CS 1005 and CS 1006. We created a web-based survey, which was given to students in several classes. We received 199 survey responses in total, including 105 students from Professor John Goulet's Linear Algebra class, and 35 students from Professor Carolann Koleci's Physics class. The survey group was self-selected; all students in the courses were asked to participate, but participation was not required. Some professors, including Goulet and Koleci, offered extra-credit of some sort in return for survey participation.

A variety of majors were represented (see Fig. A.1). For the purposes of this project, we did not consider the results of Computer Science majors or Electrical and Computer Engineering majors because their needs differ from the needs of non-majors regarding an introduction to computer science.

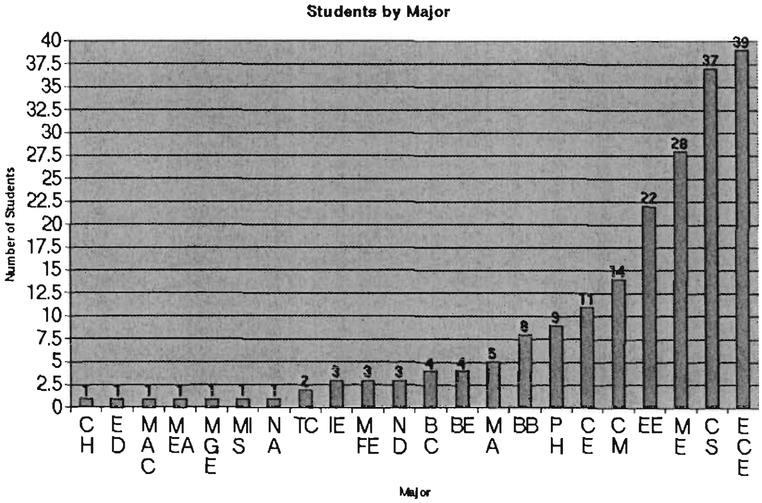


Figure A.1: Students by major

We created a number of hypotheses to test. We hypothesized that students who take CS 1005 or CS 1006 find the course too difficult and require prior programming knowledge. We also hypothesized that non-majors are taught too much CS theory. We were also seeking students' reactions to the simplicity of Python versus Scheme.

The results of our survey can be seen in its entirety below. Our hypotheses were mostly affirmed. About half of the non-majors indicated that the course they took was too hard (see Fig. A.2). About half the non-majors also indicated that they felt they had learned how to program (see Fig. A.2). Our results also showed that 67% of students preferred Python over Scheme (see Fig. A.2).

We did get some results which we didn't expect. We were surprised with the results regarding course difficulty. When the responses regarding course difficulty are plotted, the results follow the typical bell curve (see Fig. A.3). This is especially

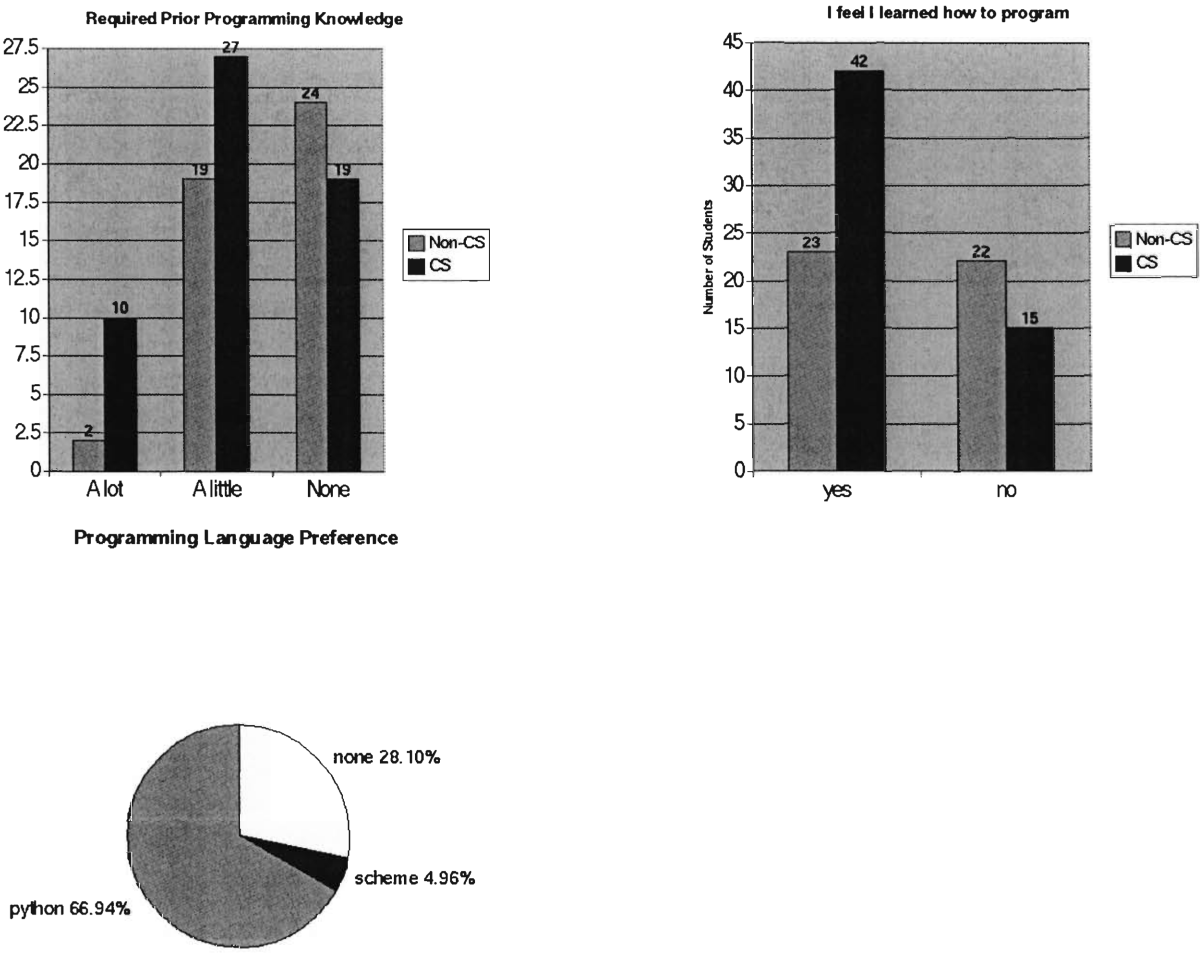


Figure A.2: Expected results

surprising because many students (about fifteen) indicated in qualitative results that the courses were difficult. We were also surprised about the results regarding the question on theory versus practical programming (see Fig. A.3). Those results also seem to contradict the qualitative data we received. About eighteen students indicated they want to see more practical applications of programming and how it relates to their major. Another unexpected result was the number of students who planned on using programming in their careers. A number of CS majors indicated they wouldn't use programming. As computer science majors, this is particularly surprising because we know first hand a career in computer science usually requires extensive programming.

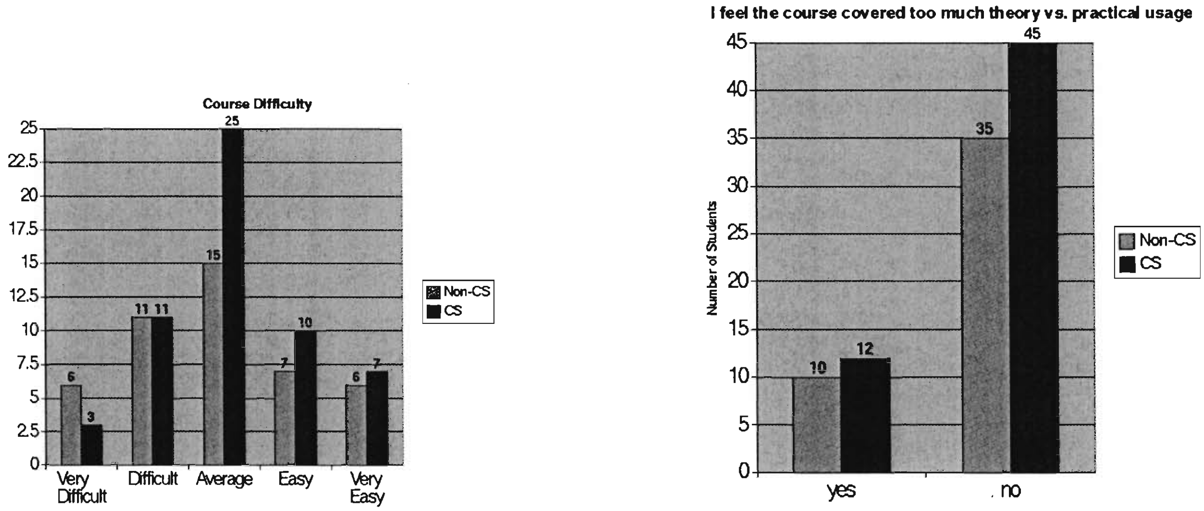


Figure A.3: Unexpected results

Overall our survey confirmed our hypotheses. The qualitative results gave us detailed information about exactly what non-majors wanted to see in a programming course. To see all the survey results, see Appendix A.1.2. The survey questions can be found in Appendix A.1.1

A.1.1 Student survey questions

IQP Survey

Intro CS classes for non-Computer-Science majors

General Information

1) Student Information:

Major: ____

Year: ____

2) If you are filling out this survey for a class, please fill out the following:

Name: ____________________

Email: ____________________

Professor: ____________________

Questions on WPI CS courses

3) Have you taken any CS courses at WPI? (If not, skip to the next section)

[_] CS 1001: Introduction to Computers (FORTRAN)

[_] CS 1005: Introduction to Programming (C++)

[_]   CS 1006: Object-Oriented Introduction to Programming (Java)

Other:

----

4) Course Difficulty: (Choose one)

[_] Very Difficult

[_] Difficult

[_] Average

[_] Easy

[_] Very Easy

5) Required prior programming knowledge: (Choose one)

[_] A lot

[_] A little

[_] None

6) Have you used the knowledge obtained from this course in any of the following?

[_] Other WPI course

[_] Project

[_] Real-world experience

If yes, please elaborate:

----

7) Do you plan on using the knowledge obtained in your future career?

(_) Yes / (_) No

8) Did you enjoy the course?

(_) Yes / (_) No

9) Do you feel you learned how to program?

(_) Yes / (_) No

10) Do you feel that taking the course was worthwhile?

(_) Yes / (_) No

11) Do you feel the course covered too much CS theory vs. practical usage?

(_) Yes / (_) No

General Questions

12) Have you done any programming other than WPI CS courses?

[_] High School

[_] Non-CS WPI course

[_] Outside of WPI/HS

13) If you have programmed in WPI courses outside of the computer science department, please list the class and language used?

----

14) If you have not taken programming course at WPI, why not?

[_] No interest

[_] Too difficult

[_] Not required

[_] No use for it

[_] Already know how to

[_] Plan to

Other:

----

15) Do you feel WPI students in non-CS majors should learn some programming?

(_) Yes / (_) No

16) What is your opinion of WPI's programming courses for non-CS majors:

----

17) What would you like to see in an introductory programming course

for non-CS majors?

----

18) General Comments:

```
Last Question:
```

```
19) Below are two languages we are considering for an introductory
computer science course. If you prefer one over the other, please mark
below:
(_) language A / (_) language B / (_) No preference
```

Language A

```
;; factorial
(define (fact x)
  (if (<= x 1) 1
      (+ x (fact (- x 1)))))


;; quadratic equation
(define (quadeq a b c)
  (let [(disc
     (sqrt (- (expt b 2) (* 4 a c))))]
    (cons (/ (+ (- b) disc) (* 2 a))
      (cons (/ (- (- b) disc) (* 2 a))’() ))))
```

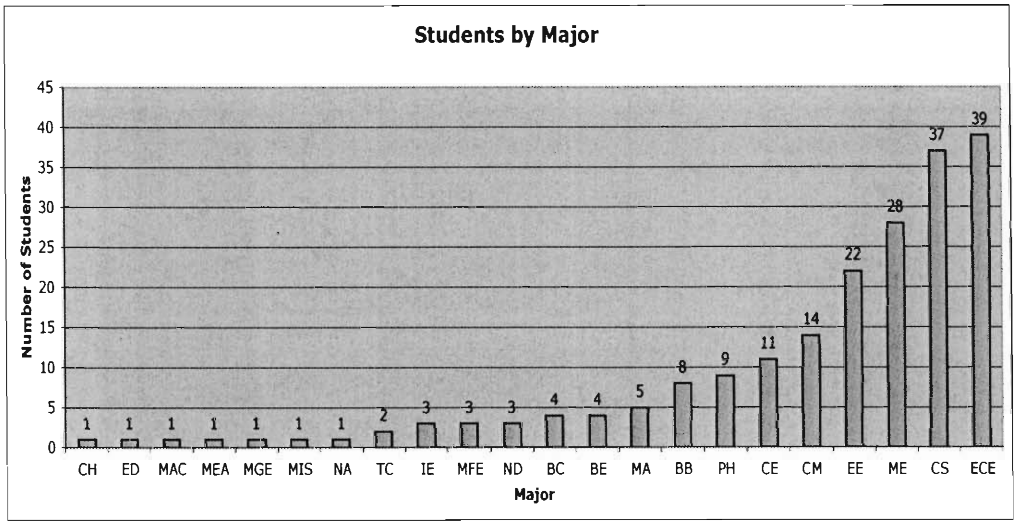
Language B

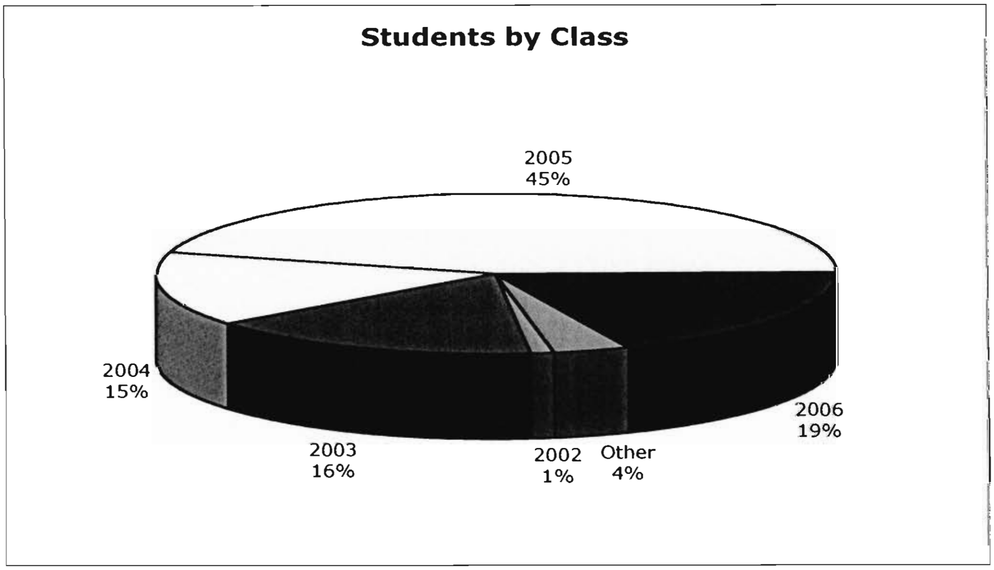
```
# factorial
def fact(x):
    if (x <= 1):
        return 1
    else:
        return fact(x-1) * x

# quadratic equation
from cmath import sqrt
def quadeq(a, b, c):
    disc = sqrt(b**2 - 4*a*c)
    ans1 = (-b + disc)/2*a
    ans2 = (-b - disc)/2*a
    return (ans1, ans2)
```

A.1.2 Student survey results

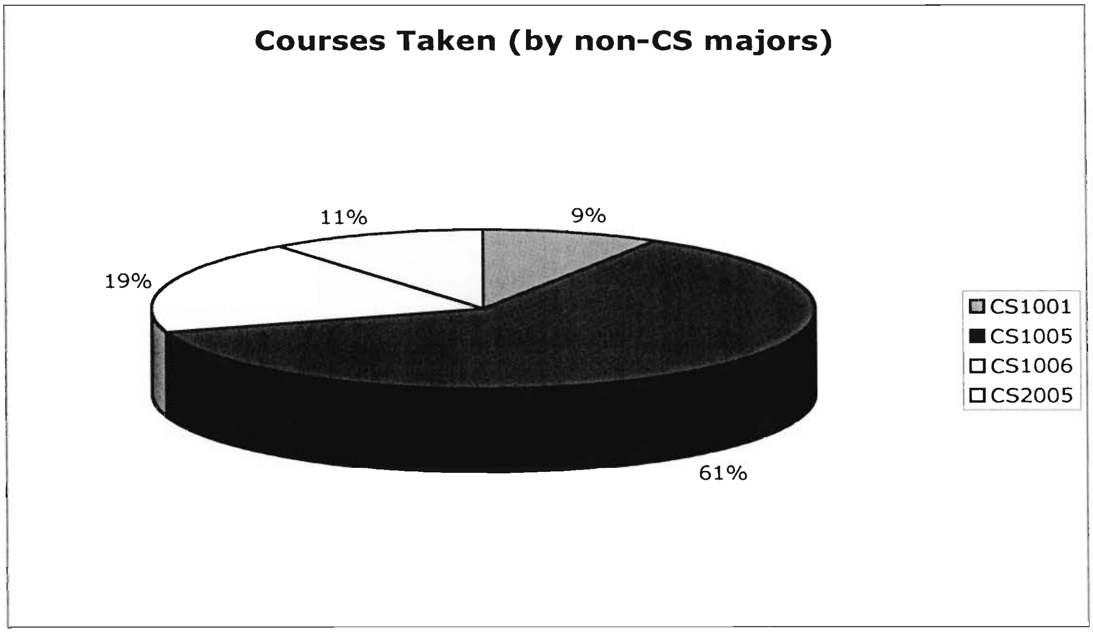
I. General Information

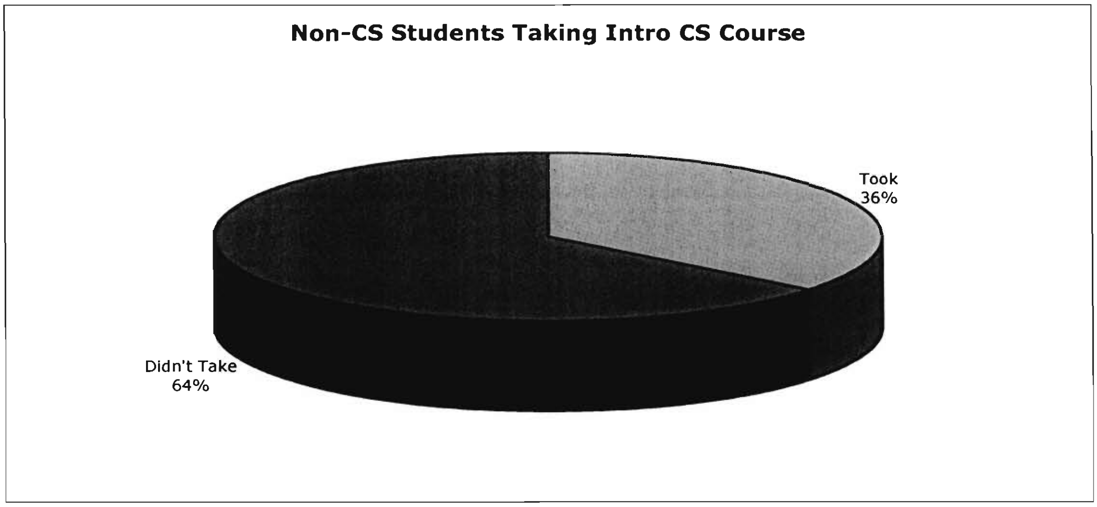


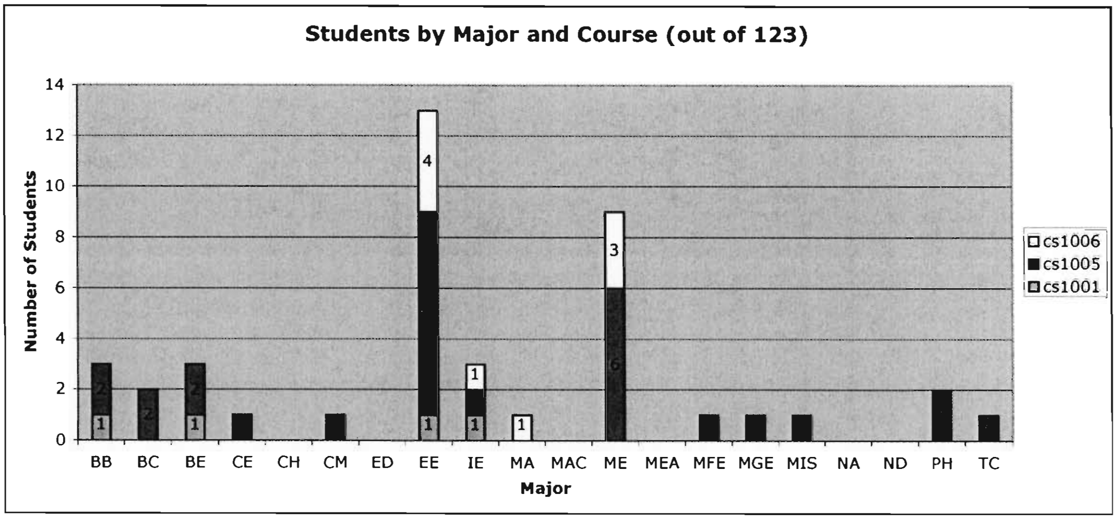


II. Questions on WPI CS Courses

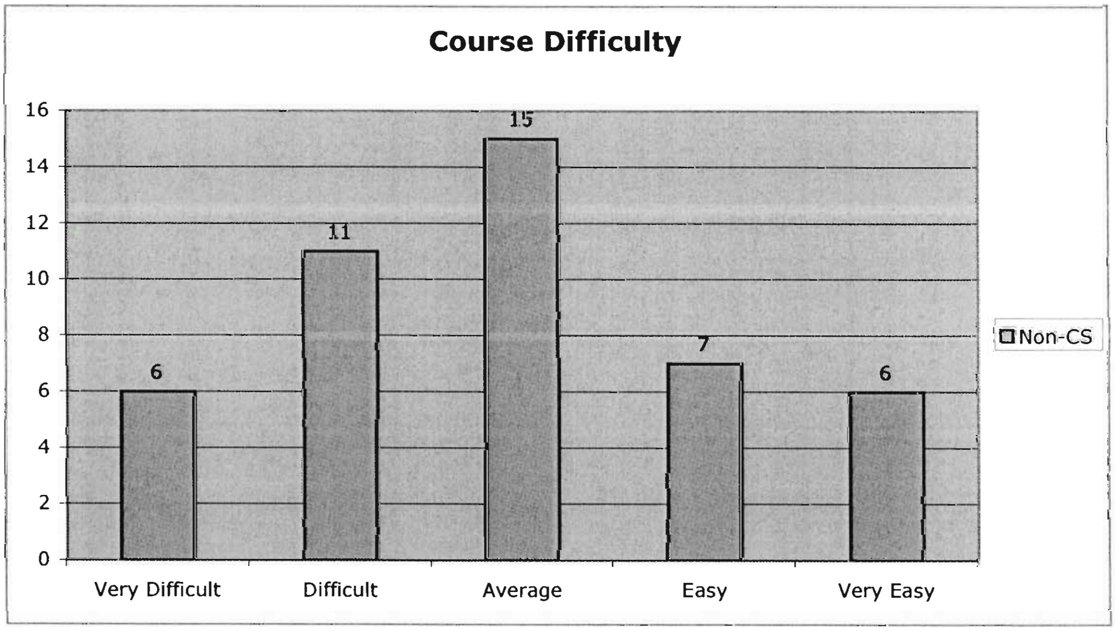
3) Have you taken any CS courses at WPI?



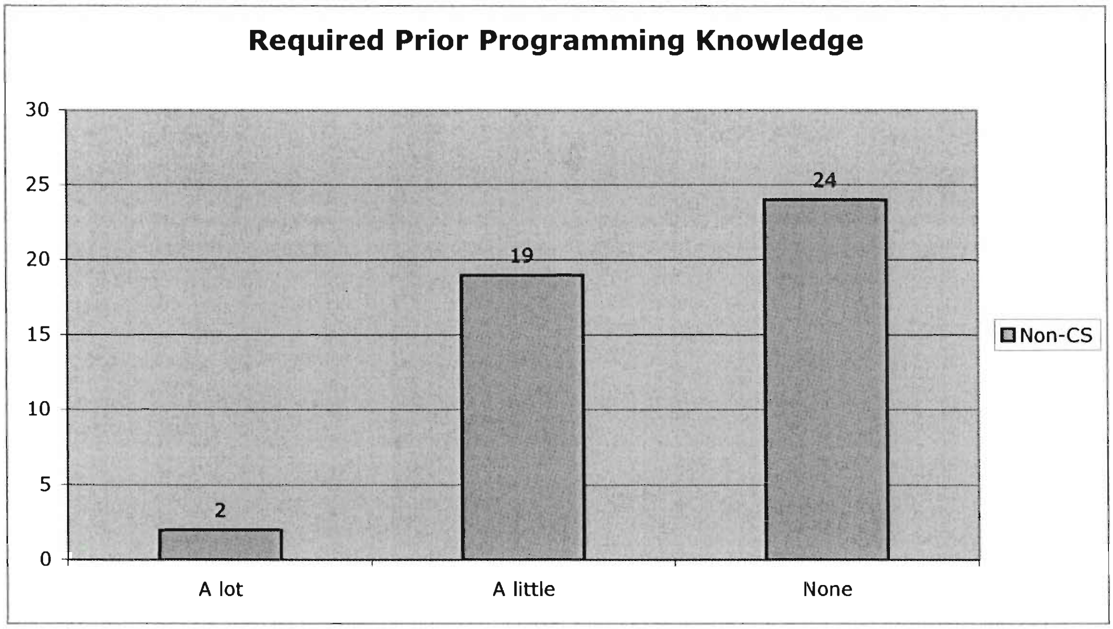




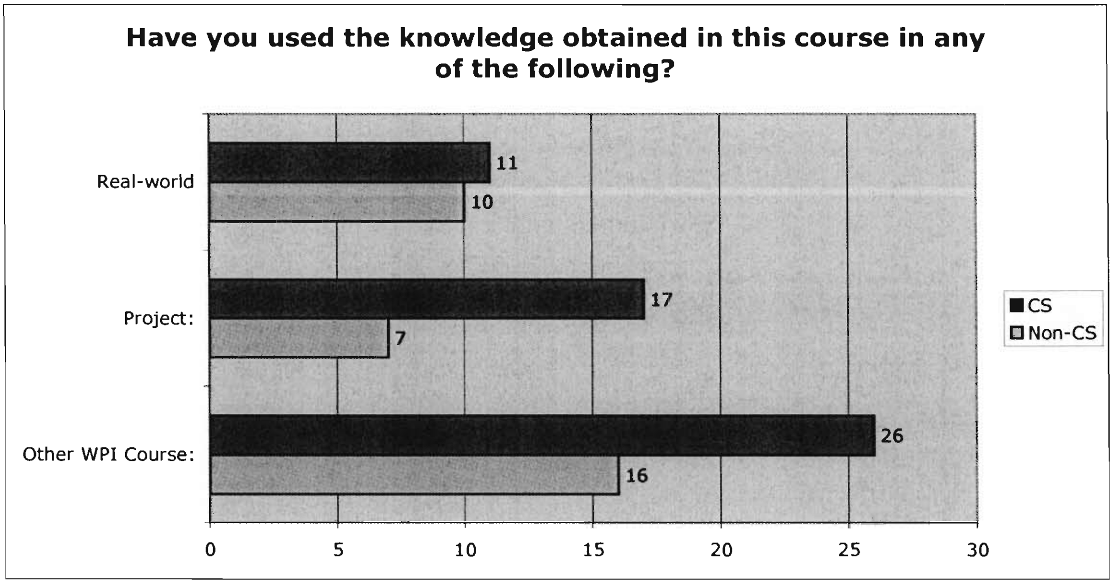
4) Course Difficulty



5) Required prior programming knowledge



6) Have you used the knowledge obtained from this course in any of the following?



Other use:

```
13  BE GS    Learning programming fundamentals helped with many other languages (IDL, Matlab,
             etc.).
15  BE 2003 MQP work
63  ME 2004 There was a need for some slight programing in EE3601 in the labs.
```

76 ME 2003 Another course in my ME/Manufacturing schedule (Robotics)requires us to learn a programing language it is similar to C++ an my prior knowledge helps.

79 ME 2004 Compuware Corp. Engineering intern

90 ND 2006 see MA 2071 below, / use programming often for problem solving and automating common tasks

100 TC 2004 I've used the concepts in 1005 to help me in other CS classes, and discrete math.

101 TC 2003 Class was about ethics of computing. Will use ethical lessons in real life.

102 EE MS I've found the concept of a semaphore is extremely useful in visualizing real world processes. The course material has also been useful to me in understanding what is happening when the unexpected occurs ... even just in everyday computer application usage. / / Looking back though, I've often thought that if I had only one course to take, CS 4533 [compilers] would have been more practical and perhaps even advance my understanding of human linguistics [I actually ended up using lex and yacc in my Music MQP to compile synthesis descriptions for instruments in a language called 'Csound'] / / Btw: I feel I've learned how to program on my own...not in an educational setting (see answer #9)

105 EE 2004 I was planning a CS minor which I have recently decided against, but I had a summer internship as a java programmer after freshman year.

106 EE 2004 This introduction to programming helped me to understand the basics of programming when I programmed in assembly in various EE classes.

109 EE 2005 internship

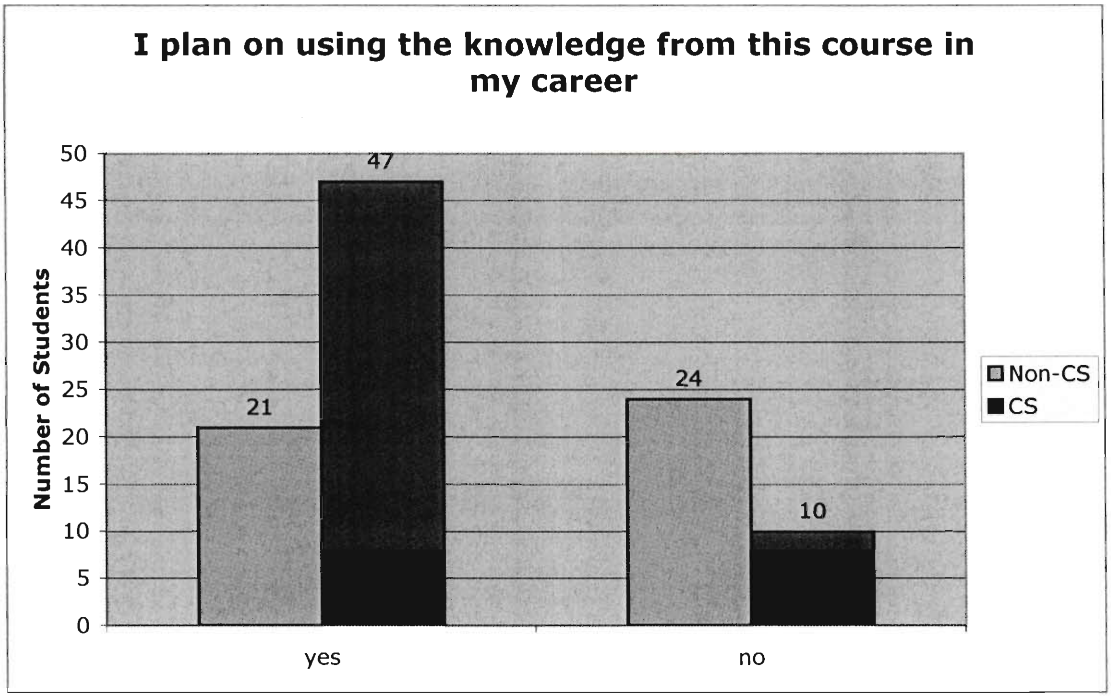
112 EE 2003 CS2005

117 EE 2004 In EE2801, Foundations of Embedded Systems, I was able to more easily visualize certain operations in C++ than in 8086 assembly, so it was of use a a sort of modeling code.

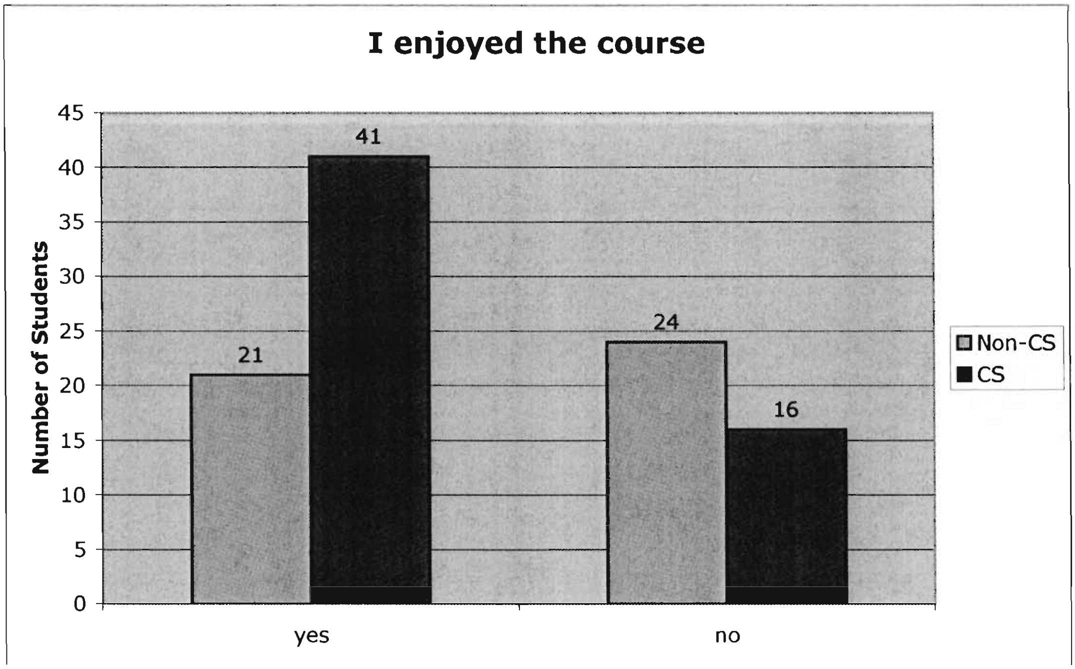
119 EE 2003 No, not yet.

121 EE 2003 Programming is cool.

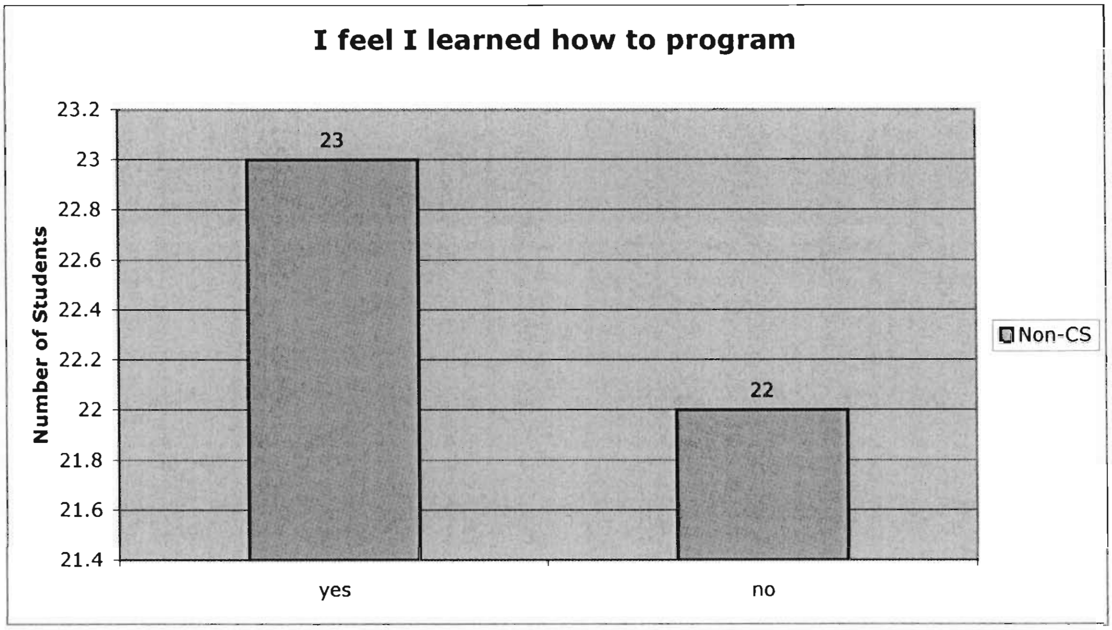
7) Do you plan on using the knowledge obtained in your future career?



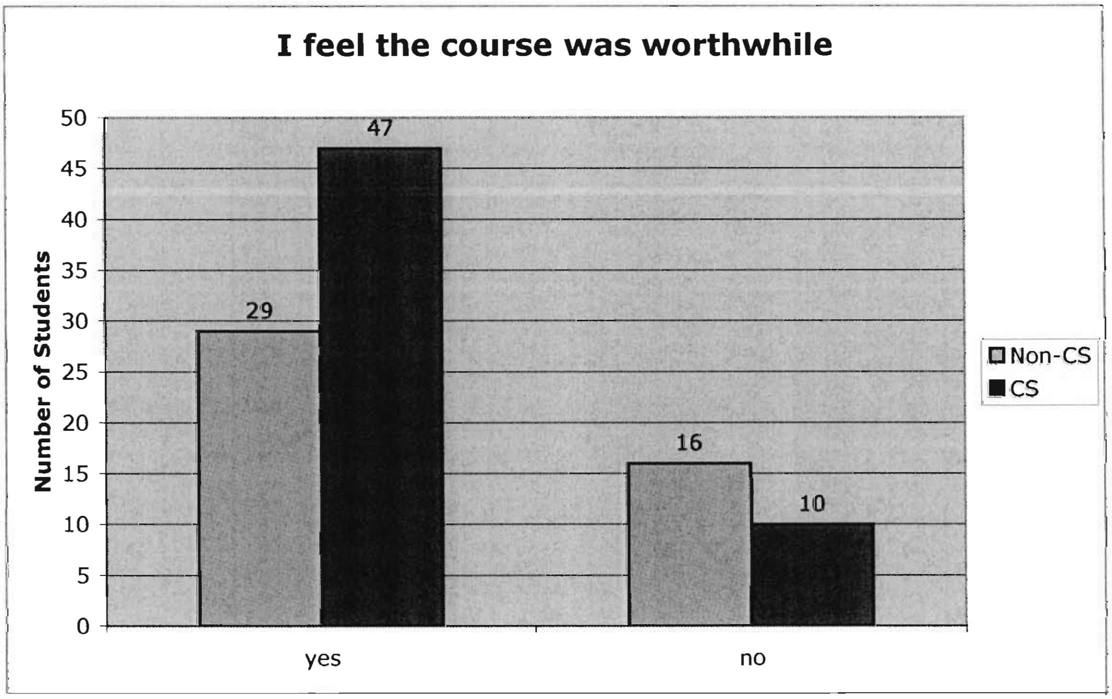
8) Did you enjoy the course?



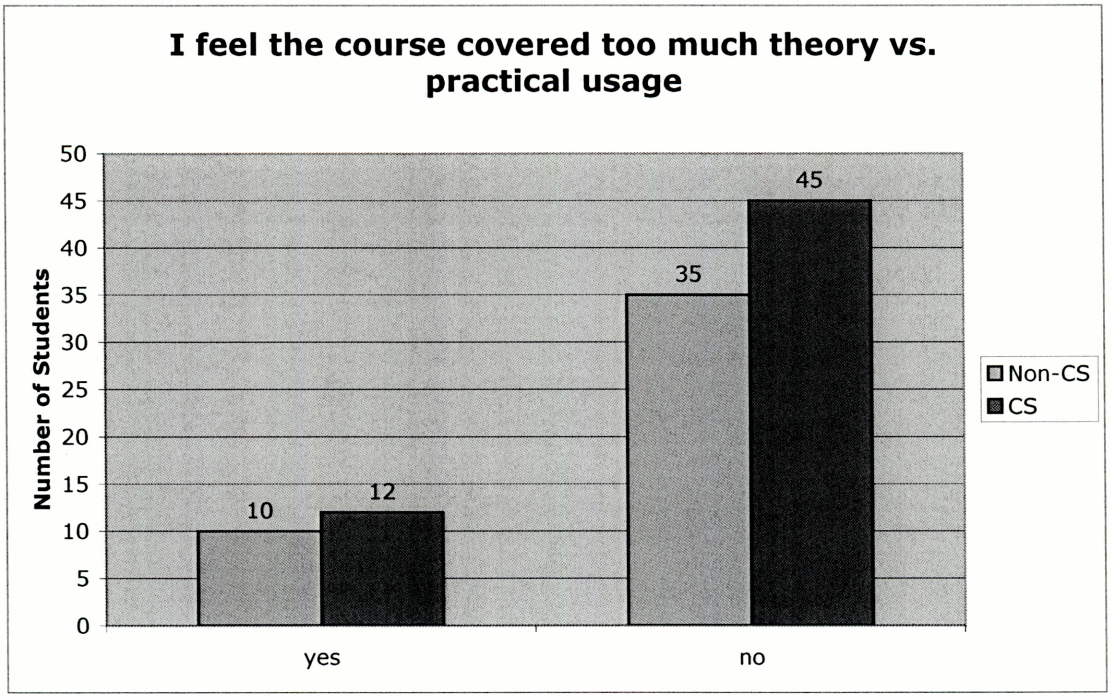
9) Do you feel you learned how to program?



10) Do you feel that taking the course was worthwhile?

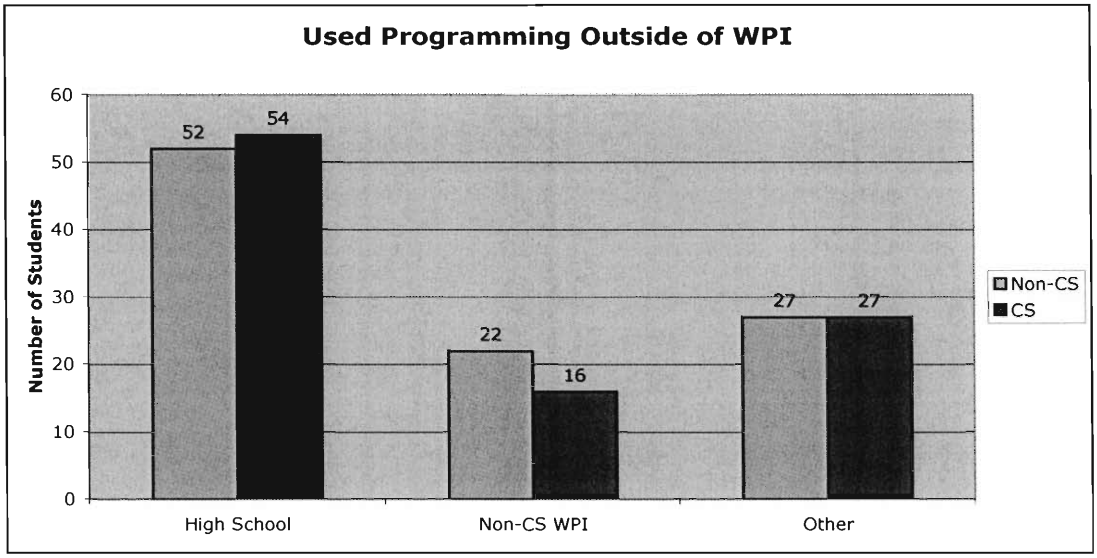


11) Do you feel the course covered too much CS theory vs. practical usage?



III. General Questions

12) Have you done any programming other than WPI CS courses?

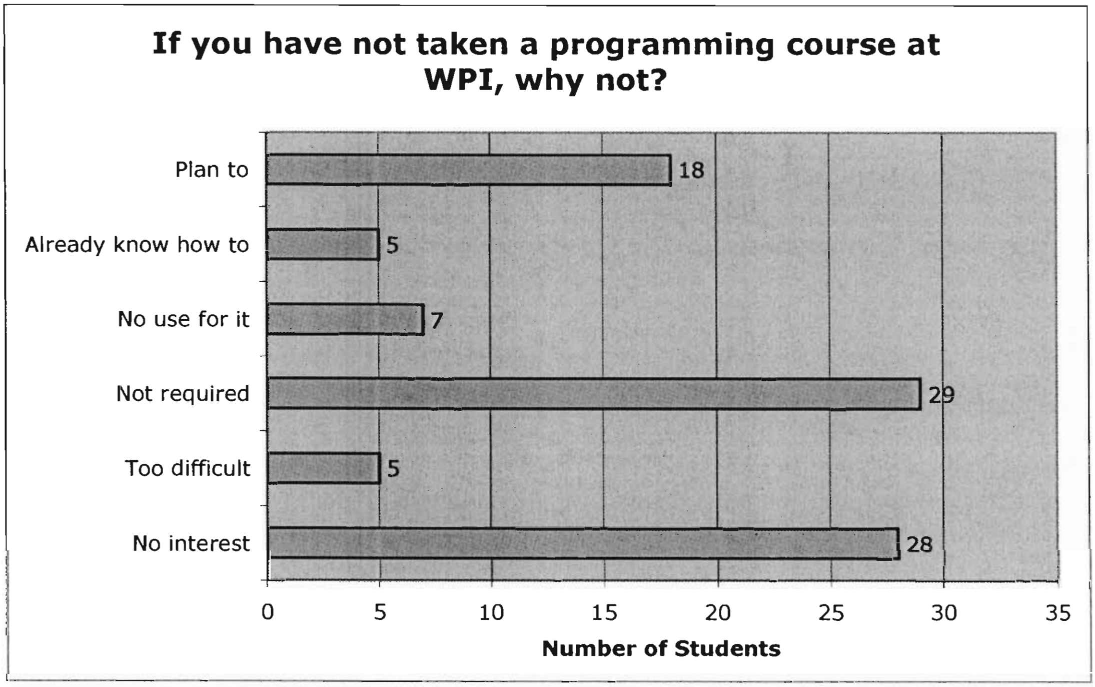


13) If you have programmed in WPI courses outside of the computer science department, please list the class and language used.

```
5   BBT      2003  Calculus-Maple
8   BBT/TC   2004  gwbasic, visual basic
12  BC 2003 Maple
13  BE GS    BE4201 - IDL / Many courses - Matlab
16  BE 2003 BE4201-IDL / EE3815-VHDL / EE2801-Assembly
17  CE 2005 i havent programmed at all
20  CE 2003 C+ / Visual Basic
21  CE 2004 Introduction to computing, C++
27  CE 2003 no
32  CM 2003 BASICA - individual study
38  CM 2004 Chemical Engineering Thermodynamics, used MathCAD, which contains the same
            systematic approach and troubleshooting ability of computer programming.
44  IE 2003 MG 2720, Visual Basic
46  IE/ME    2004  MG2720 - Visual Basic
50  MA 2006 C++
51  MA/ME    2004  Numerical Methods for ODE's. Matlab / Control Engineering. Matlab
58  ME 2006 G-CODE IN ME
61  ME 2005 C++ Computer Programming
65  ME 2006 Intro to C / C++
74  ME 2005 HTML...
76  ME 2003 Robotics- VAL-II
77  ME 2004 VBA Basic
78  ME 2005 Intro to computer programming: Microsoft Visual Basic
79  ME 2004 ME2300, C
90  ND 2006 MA 2071: Linear Algebra; used C++
92  PH 2006 Introduction to computer science course.  Just for one year.  C++ was used.
100 TC 2004 In HS, AP Computer Science: / Karel the robot / C++
102 EE MS   EE503 - Digital Signal Processing / EE539A - Real Time Digital Signal Processing /
            Music MQP / EE514 - Fund. of Radio Freq and Microwave Engineering / Probably many
            others.
103 EE 2006 Sophomore year of high school I took an intermediate VB course at the local
            community college.  The intro course was too easy, but the intermediate was a
```

			little above my head. / In high school, a VERY introductory course to programming using qbasic. I didn't learn much, as I had picked up a little C by then.
105	EE	2004	Assembly language programming for EE2801,EE2799,EE3803
106	EE	2004	EE2801 - assembly / EE3803 - assembly
111	EE	2005	EE 2801, assembly language programming
112	EE	2003	EE 2801 Intel/PIC Assembly / EE 3803 Intel/PIC Assembly / EE 4801 Intel/PIC Assemply
116	EE	2003	EE2801 Assembly
117	EE	2004	EE2801 - x8086 Assembly
118	EE	2006	AP Computer Science in high school - C++
119	EE	2003	EE2311 (Signals) Matlab / MA2071 (Linear Algebra) Matlab / PH1140 (Waves and Osc.) Matlab for graphing lab data
120	EE	2004	Pascal, ASM, C/C++, VB
121	EE	2003	EEx8xx courses

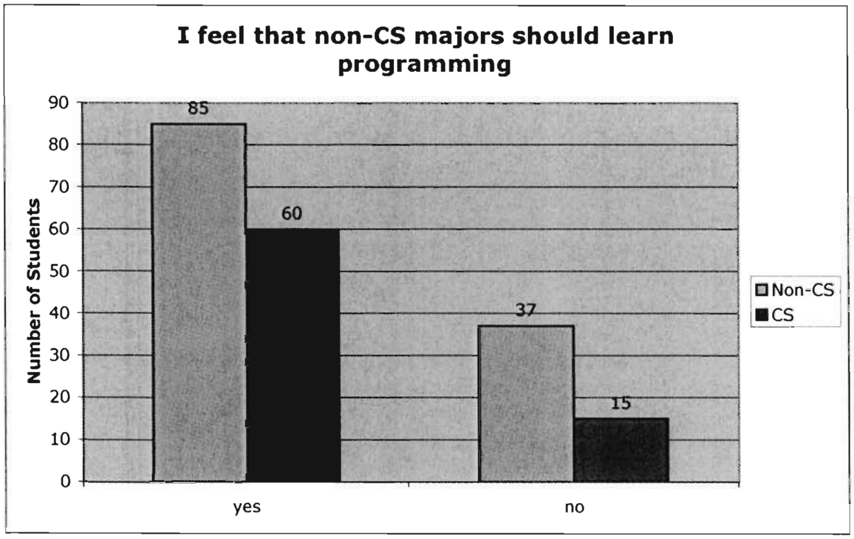
14) If you have not taken programming course at WPI, why not?



Other:

12	BC	2003	didn't fit into my schedule
21	CE	2004	I just transferred here in A-term.
24	CE	2006	I would like to learn programming but I don't think I have room for it with all the other classes I am taking.
32	CM	2003	not required/scheduling conflict
35	CM	2003	have not really had the time /
38	CM	2004	I am graduating early, so my course schedule is as compressed as it can be, so I do not have time for extra electives. It would have had to take the place of something else.
53	ME	2006	I have not had time in my schedual yet.
60	ME	2005	No time
77	ME	2004	Scheduling, availability
90	ND	2006	Programming can be learned without taking a whole course about it.
92	PH	2006	I plan to study by my own. I just want to know the basic stuff, so I do not think I have to take it as a class.
102	EE	MS	I avoided the CS1000 and 2000 series because I'd prefer to learn theory than yet-another-programming-language.
103	EE	2006	I want to work with embedded systems and circuit design as an EE, but programming isn't an interest of mine. I've learned some C and VB on my own, and I'm going to take 3-4 total CS courses while at WPI.
119	EE	2003	I'm a transfer student close to graduation, and I only had time in my schedule for one CS course. Also, a 2000 level course is the only degree requirement. Otherwise I might have taken C++ (CS1005 -> CS2005)
123	EE	2004	I have taken programming courses at another university.
124	ECE	2006	Have not been able to fit into my schedule yet.
132	ECE	2005	I plan to take the ECE assembler course, but probably no courses in CS.
135	ECE	2005	In addtion to my one course I'd like to take more but can't find enough time.
167	CS	2006	And I'm taking one now

15) Do you feel WPI students in non-CS majors should learn some programming?



16) What is your opinion of WPI's programming courses for non-CS majors?

1 BB 2005 I don't know, I haven't taken any yet

3 BBI 2003 They are hard

5 BBT 2003 They are either too easy or too hard.

7 BBT 2004 I have never taken one.

8 BBT/TC 2004 There really are no 'CS for non majors' classes

9 BC 2005 I have not taken a CS class myself, but I have heard from other students that they are difficult if you have never taken programming before.

11 BC 2003 They are non-existent

15 BE 2003 Haven't taken any, but seem like a good idea.

16 BE 2003 The lower level courses are broad enough to have useful applications for all majors.

17 CE 2005 I have no opinion because i have not taken any of these courses

18 CE 2005 I believe that for some courses, they make sense, but the vast majority are unnecessary. For example, I don't think it's necessary for calculus 1-4 required to use Maple.

19 CE 2005 I don't know enough about the pogramming courses for non-CS majors

20 CE 2003 I think that there should be a Computer Science Course offered for all other majors similar to The EE3601, which is an Electrical Engineering course for non Electrical Engineering majors.

21 CE 2004 Unless other majors are going to implement it in some way I do not feel it necessary to take it.

22 CE 2005 Have never taken any.

23 CE 2005 I don't know anything about the programming courses for non-CS majors.

24 CE 2006 I don't know anything about them, so I couldn't say.

25 CE 2005 I have no knowledge on this question.

26 CE 2005 I didn't know there were programming courses for non- CS majors.

27 CE 2003 i don;t think is a good idea

28 CH 2003 I have heard that the introductory courses assume a basic knowledge of programming. Since I have never taking any type of CS course, i have shied away from the ones at this school

29 CM 2006 Quite helpful.

30 CM 2003 I haven't taken any, so I don't really know.

31 CM 2004 I do not know of the non-CS course, only of CS 1005

33 CM 2003 too difficult

35 CM 2003 don't really have one /

38 CM 2004 I think it would help to be able to set up basic computer programs that would allow me to set up iterative loops to solve complicated problems.

42 CM 2003 I know nothing about cs courses except they are considered hard

43 ED 2006 I have no opinion as of now because I have not taken any non-CS programming courses and have not heard much about them.

44 IE 2003 Too intensive, should be taught slower.

45 IE 2003 the basic courses seem to cover enough

47 MA 2006 Haven't seen them

48 MA 2006 They could have some more basic options.

52 MAC 2006 I feel that they everyone should take CS courses due to our highly technological society. I don't know of the courses in general, but I feel that they are very necessary.

54 ME 2006 I haven't looked into it, but i will be taking intro to CAD, not that that counts or anything

56 ME 2006 Let it be known that I didn't know that there were non-CS major programming classes, nor do I know anyone who has taken any such classes.

58 ME 2006 EASY ENOUGH FOR NON-MAJORS TO CONSIDER TAKING, AND INFORMITIVE ENOUGH FOR THEM TO FIND THEM WORTH TAKING.

62 ME 2004 WPI allows students to specialize in their specific topics of interest. I don't feel that early programming classes fit into most non-CS fields.

63 ME 2004 Very few offered, and mainly geared towards CS majors or people with CS experience.

64 ME 2005 I don't know much about them, I've never taken one. I'm not sure that I will unless I have to or can see that it will have a direct/major impact on my career goals.

65 ME 2006 Not taken any yet

66 ME 2006 Courses in CS should be required, so long as they are applicable to other majors.

67 ME 2005 Few and far between

68 ME 2004 The CS classes help you understand the process behind programs but I don't think you should have to take CS classes if you are not a CS major.

69 ME 2005 haven't taken any

70 ME 2006 Don't know much about it.

71 ME 2005 I have no experience with them

72 ME 2005 It would be good however for all students to have some knowledge of programming.

73 ME 2005 It would be good however for all students to have some knowledge of programming.

74 ME 2005 I dont really know much about it

75 ME 2005 Never taken one

76 ME 2003 They should at least learn the basics since the increased need for programing today.

77 ME 2004 no idea what they are like

78 ME 2005 Have not taken a programming course yet, but I assume very good.

79 ME 2004 No real world applications, therefore irrelevant. /

80 ME 2003 I thought for a beginner class that it was very difficult to grasp the concept of programming.

81 MEA 2005 I haven't taken any CS courses myself, but I've heard from non-CS majors who've taken introductory CS classes that they are very difficult if you have no prior programming knowledge.

82 MFE 2006 No idea really, I will tell you after I take 1005 in C Term.

83 MFE If you don't have any basic knowledge of programming, it'll be pretty hard for you to start 1005

84 MFE 2005 I feel they are unnecessary

85 MGE 2003 I think that there is no option for wpi non CS majors. Fortran is basically a dead language, and the C++ class is a CS class that CS majors have to take. So the non CS majors might need the class to go slower or less in depth and the basic concepts enforced more, while the CS majors may get bored. there should be two C++ classes-one for majors and one for non majors

86 MIS 2004 They should be available but not required.

87 NA NA I don't know that much about them.

88 ND 2006 They seem to be difficult for some people with bad teachers, that make it more beneficial to learn from the book than go to class.

89 ND 2006 I am not sure because I hasven't taken it but mt roommate says the classes are very hard and she is nonCS.

90 ND 2006 useless, unless student is too lazy to learn programming on his/her own

91 PH 2005 When I took 1005 (A02) the course was mostly theory. I enjoyed learning about it, but the class grades were on programming. Since I did not know how to program or organize an idea to start programming I had alot of difficulty in the class. Those CS majors who already knew the language had an easy time because they only needed to know the language and what was taught in class was almost irrevelant.

92 PH 2006 I want them to offer courses useful to my interest of study.

93 PH 2006 They should not be programming language specific. One may be focused on, but it should not be all htat is taught.

94 PH 2005 no opinion

95 PH 2005 Don't know. Haven't taken any.

97 PH 2006 No opinion, I haven't experienced them yet.

99 PH 2005 Don't really have any experience on the subject

100 TC 2004 They are apt, in that an introductory programming course in any language will generally teach the basic concepts of programming on its own.

101 TC 2003 have not taken any

102 EE MS Everyone should learn how to write some code. It changes the way you think... just like engineering courses.

104 EE 2005 I am sure they will be the same caliber as that the CS majors take.

105 EE 2004 NA

106 EE 2004 I have not taken one yet, so I'm not sure.

107 EE 2005 That they are relatively easy and you get what you put into them, just like any other class.

108 EE 2003 i haven't found any classes meant for non-CS majors. one or two courses teaching practical programming techniques with minimal theory would be very useful.

109 EE 2005 i dont know, I havent taken them

110 EE 2005 CS 2005 seems pretty difficult for non-cs majors, and easier course would be better

111 EE 2005 No too bad, needed for EE.

113 EE 2005 There really aren't any because if you take a cs course you are mixed with cs majors

116 EE 2003 I think they should have a seperate course for people not majoring in CS.

117 EE 2004 I think that they are okay, so far. I think that cs1006 has a much sharper learning curve that 1005 though.

119 EE 2003 N/A

120 EE 2004 They are good.

121 EE 2003 good. should have a visual C course. No cs (like EEs) need C but need GUI's for projects and class

123 EE 2004 Programming courses are useful for the future of designing software. It is always good to know where it comes from.

125 ECE 2005 I have not taken programming here as of yet, but I feel from reading the course overviews that the 1001 and 1005 level courses should be enough to give even a History Major a good ideo of how to design simple programs. Plus the students leave the course with a better idea of what computers are and how they do what we want them to.

126 ECE 2004 Non CS majors shouldn't be required to take CS courses. If they want to, they can. /

127 ECE 2004 It's not very useful due to it using only C++ but it is useful if you realize that sometime in the future you will need to use C or C++ and it does teach you how to program.

128 ECE 2004 Horrible /

129 ECE 2006 i dont really know

130 ECE 2006 I would not know,I plan on taking a programming course nexy year.

131 ECE 2006 It can be useful, but if you're not going to be using it ever I don't see any reason in taking a course.

132 ECE 2005 I do not really know much about them, though I have heard that the ECE assembler course (2801?) is important to take.

133 ECE 2005 I didn't know they existed specifically.

134 ECE 2005 Never taken them

135 ECE 2005 I think they are helping and informative.

136 ECE 2005 I think the courses have a good layout, but i have yet to take a course in CS yet.

137 ECE TR Useful knoledge

138 ECE 2005 it can be useful if taught well

139 ECE 2005 It is very helpful for many of the other majors because it helps you understand how to sometimes use other software. One example is when I started using matlab, my background in programming made it much easier to learn how to use that software.

140 ECE 2005 Good

141 ECE 2005 CS 2005 is much too difficult for non-CS majors. It is also not very usefull outside of the CS major.

142 ECE 2005 if you need it for your major then take it

143 ECE 2005 Adequate.

145 ECE 2005 All I know from experience is about CS1005. I feel it was a worth while course and everyone should take it. At some point, all engineers need to do some type of programming, usually Matlab.

146 ECE 2006 For some majors it is needed but it really depends on the major

147 ECE 2005 Not sure havent taken any yet

149 ECE 2005 Introductary courses such as cs1005 provide good general knowledge and background.

150 ECE 2005 They expect too much, from the homeworks especially. And CS1005 is supposed to prepare you for other programming courses: But so far, in CS2005, I haven't seen anything that I would be unable to do without the knowledge I have from CS1005.

152 ECE 2005 I feel that all students should become familiar with at least miniumal programming.

153 ECE 2002 Havent taken them yet

154 ECE 2005 Programming is a way of thinking, it can help anyone in any major.

155 ECE 2005 I have not taken them.

156 ECE 2005 they have some use especially in some of my ECE cources

157 ECE 2005 run! RUUUUUNNN!!! i mean.. 1005 and 1006 are pretty easy for anyone, but after that, it's all downhill...

158 ECE 2005 I don't think it is necessary.

159 ECE 2005 Good

160 ECE 2005 Need classes in Matlab

162 ECE/HU 2005 Good

163 CS 2006 Being a CS major who hasn't taken an introductory CS course here, I don't really have an opinion on that.

164 CS 2006 The CS 1005 course didn't really have much theory, more just how to use the commands. It wouldn't help if they wanted to use other languages than C++. If

they take it, they'll just get credit for the course and not much else will happen since it doesn't detail how to apply it in everyday life, really.

165 CS 2005 Assumed a little bit too much prior programming experience, but still a good intro.

166 CS 2005 Provide a good background in programming which could come in useful in other profesions.

167 CS 2006 Not sure what is required now

168 CS 2004 There should be a couse in CS similar to Volts-for-Dolts in the EE dept.

169 CS 200? The current state is optimal

170 CS 2006 There aren't many because they start right in with programming they need to take a more basic appraoch.

171 CS 2005 good

172 CS 2005 If people have never programmed before or have no experience with computers, I think it would be very hard for those people to learn a language a 7-week term.

174 CS 2006 They should be optional.

175 CS 2005 too easy.

176 CS 2003 From what I know, they are taught as if the students are CS majors

177 CS 2005 I think if i had not had any background of programming the introductory courses would have been pretty hard.

178 CS 2005 They are easy and a good background

179 CS 2005 may be a little too in depth for what they want/need

182 CS I think it is useful knowledge that people should know a little about no matter what field they intend to go into

183 CS 2006 From secondhand experience, they seem to be well designed.

184 CS 2005 It's good enough, if students are interested enough they should look into minoring

185 CS 2005 easy enough

186 CS 2005 It at least teaches you a language. That bare minimum is all you need to amaze friends and family alike with how much of a computer whiz you are.

187 CS 2005 generally a good idea

189 CS 2005 Some of them are a bit difficult for those who have had no prior programming experience and have no innate skill at it.

190 CS 2005 Upper level courses are more interesting

191 CS 2005 No opinion.

192 CS 2005 Too hard

193 CS 2005 They can be hard and uninteresting. They teach problem solving, and logic, and can be useful for non-CS majors.

194 CS 2005 They're pretty basic and good for beginners.

195 CS 2004 Many find it very difficult

198 CS/EE 2005 I think it would be good for students to know all the basics in programming within their field, or how programs can be used to get solutions easily for certain problems.

199 CS/EE 2004 I really learned how to program in EE2801 with Professor Michaelson. I have applied the top-down approach to program design in assembly to EVERYTHING that I have done since. It was probably the SINGLE class that taught me how to program well.

17) What would you like to see in an introductory programming course for non-CS majors?

1	BB	2005	Some practical applications of basic programs
3	BBI	2003	An easy introduction into coding
5	BBT	2003	Yes
6	BBT	2005	smaller classes and no ACTUAL required knowledge
7	BBT	2004	Yes, definitely. I feel that a very basic course teaching a beginning level of programming for non-CS majors would be helpful for students in almost any field.
8	BBT/TC	2004	I think for those without prior exposure to any programming and without much further use for it should be giving the chance to at least get a basic knowledge
9	BC	2005	Very, very basic introduction designed for people like me whose computer knowledge extends to typing up papers.
11	BC	2003	practical application for future courses (non-CS)
12	BC	2003	yes, it would be helpful, even a basic course in how to use UNIX, Linux, ot something else. At least in the field of x-ray crystalography, the software is run on UNIX base, which is foreign to myself and most others.
13	BE	GS	i thought that's what cs1001 is
15	BE	2003	how to create algorithms, programming basics not particular to a specific language.
16	BE	2003	Um, CS 1005 and CS 2005 are good enough. We don't need a 'special' non-CS programming class. Get real.
17	CE	2005	i dont know anything about programming
18	CE	2005	I believe that all the different departments should develop their own CS classes that are designed to teach the necessary programming required for their major.
19	CE	2005	Introduction to some sort of basic language, or application of Access for basic programs
20	CE	2003	A course that uses material from other majors related to computer science.
21	CE	2004	Fortran.
22	CE	2005	Dont jump right into it, it would be hard for some people to understand the concepts at first, so dont turn them off right away.
23	CE	2005	I don't think they are necessary for non-CS majors unless you really want to learn. I am not interested so I don't care what is in the course.
24	CE	2006	I don't know. I would like to see the same stuff that is in an introductory course for CS majors. I wouldn't want a watered down course although I way enjoy a course that extends the applications of programming into, perhaps, the field I am studying. Maybe Introductory Programming in the Civil Engineering Field. hehe
25	CE	2005	I would like to see students get a good understanding of the basics functions involved in the most popular languages.
26	CE	2005	The basics, how to do something and why.
27	CE	2003	some essy things.
28	CH	2003	The 'assumed' Knowledge should be taught
29	CM	2006	I would like to get a good foundation for everything... being able to, theoretically, program anything i might need to.
30	CM	2003	THe basics-I know nothing about programming.
31	CM	2004	A very basic introduction to programing, how it can be applied to other engineering majors. I would like to see more of computer use instead of programing. Very applicaple to my major
32	CM	2003	Sure.
33	CM	2003	the basics, i have no background what so ever, and i have never looked into what the introductory courses are offering now
34	CM	2005	IT'd be nice.
36	CM	2004	web page making stuff. what is called? html?
38	CM	2004	Yes, I would like to see that (although it is too late for me).
41	CM	2005	applications in non cs courses with the programming language that those majors might use in the future.
42	CM	2003	a real introduction, right from the begining... don't assume people know anything
43	ED	2006	General programming languages that can be used outside of a CS environment. No complex languages that would not be used in general situations, but something that is relatively simple yet easy to apply in real life.
44	IE	2003	Start very basic and very slow. Assume the syudent knows nothing about programming.
45	IE	2003	same things as are covered in CS1005 and CS2005
47	MA	2006	Very general course, designed more to introduce the concepts rather than just one language, so people whe take it can easily learn any other language that their major might require.
48	MA	2006	slower pace, more basic topics

52 MAC 2006 NA

53 ME 2006 A diverse set of languages used.

54 ME 2006 no

56 ME 2006 Coverage of basic programming techniques and the logic required to be a succesful programmer. Overall stress on practicality, that is a course that covers necesities for basic programming.

58 ME 2006 BASIC CS CONCEPTS TO ALLOW NON-CS MAJORS TO COMMUNICATE WITH CS-MAJORS AND MAKE THE CONNECTIONS BETWEEN HARDWARE AND SOFTWARE PEOPLE BETTER. ALSO, BEING ABLE TO WRITE SIMPLE PROGRAMS IS A BENEFICIAL SKILL TO HAVE.

59 ME 2005 a

60 ME 2005 yes

61 ME 2005 You shouldn't have to know programming unless you are going to use it in your job, pro-E is useful for ME's programming the lathe's etc, which should be taught but programming launguages such as C++ should not be necessary.

62 ME 2004 I think CS1005 is pretty standard for getting the CS idea across.

64 ME 2005 That might be nice, particularly for people like me who really don't know anything.

65 ME 2006 an intro to C++ that covers a general basis of the program

66 ME 2006 If the resources exist, it would be nice to have seperate sections for the different majors so that students can apply their CS knowledge to their area of study.

67 ME 2005 How to create things that can help in the computer related business field

68 ME 2004 A better explaination when it comes to the labs. I took cs1005 and had a lot of trouble because the teacher and the book did not explain how to program. Seeing that I am a ME major programing is hard for me to pick up on.

71 ME 2005 i dont know anything about programming

72 ME 2005 I don't really know anything about college level programming. Therefore, I'm not in a position to make any suggestions.

73 ME 2005 I don't really know anything about college level programming. Therefore, I'm not in a position to make any suggestions.

74 ME 2005 general programming

75 ME 2005 Don't know

76 ME 2003 More theory since the skill to learn different programing languages will probably be more important than one particular.

77 ME 2004 Code for Excel and Pro/E. Those are what my major needs in everyday courses.

78 ME 2005 Basic skills

79 ME 2004 Real world situations, problems, etc.

80 ME 2003 yes

82 MFE 2006 Yes, definitely. I wasn't quite sure which CS course to take as an introduction. Luckily, my roommate is CS and explained everything to me.

83 MFE Professors should spend more time on teaching how to code, not only the concepts.

84 MFE 2005 Nothing really

85 MGE 2003 The programming part to be gone through better...Its the only class I NRed at WPI and I had to retake it...and I have a high GPA. The class goes too fast for those who know nothing about programming and makes those who dont know anything about it feel like idiots

86 MIS 2004 The basics: what programming is, how to use it, etc., so students can decide if they have any interest in the subject before taking another class.

87 NA NA From what I've heard, the introductory courses are already easy enough.

88 ND 2006 Basics covered well, A nice easy pace

89 ND 2006 Yes

90 ND 2006 general programming concepts instead of details of a perticular language, although a perticular language may serve as an example of those concepts

91 PH 2005 I would like to see an introductory theory course and a separate introductory programming course.

92 PH 2006 Basic, but not too easy.

93 PH 2006 Examples of programming used in jobs from other majors.

95 PH 2005 The basic fundamentals like functions and variables and whatnot. Also, maybe the C syntax.

97 PH 2006 Mostly concepts that often come back, so as to make a student more familiar with what programming often entails, fundamentally.

99 PH 2005 a basic overview of a couple languages

100 TC 2004 The programming courses should provide an understanding of how programs generally work. It should relate directly to problems abstracted from real life, such as a way to store addresses.

101 TC 2003 yes

102 EE MS Perhaps connections to thier major?

103 EE 2006 A lot of people have very different backgrounds in programming, so it is hard to know where the intro course should start. I would like to see the first few weeks broken up into small sections, so that everyone can learn at a different speed. I know that I can't take CS 2005, but I expect to not learn anything in the first 2-3 weeks of CS 1005/6.

104 EE 2005 I would like to see every kind of programming language touched on and taught. just in case we run into something, we can have a slight idea what the CS majors do.

105 EE 2004 I think it would be beneficial to almost all non-majors

106 EE 2004 That could be helpful for those students who do wish to learn a little programming, which might help them out later on during their WPI career.

107 EE 2005 Nothing, from my experience, it was what I completely expected.

108 EE 2003 definitely

109 EE 2005 yes, if it will count for a cs credit for EE

110 EE 2005 at least some c++, and maybe a little java as well

111 EE 2005 Possibly, if it wasnt too difficult

113 EE 2005 yes

115 EE 2005 YES

116 EE 2003 yes

117 EE 2004 Less theory, more problem solving exercises.

119 EE 2003 I'm not sure that I can comment on this directly, because I came to WPI having some prior programming experience. I went straight into EE3815 (VHDL), which as a 'prereq' recommended some general programming experience. So I think the intermediate level is good for a non-CS major. I think Verilog would be very useful to an EE.

120 EE 2004 No...no need

121 EE 2003 GUIes

124 ECE 2006 Show practical usage of programming in areas beyond CS.

125 ECE 2005 A small section that (couple of days of course work) that explores the inards of a computer. Most graduates these days, if not all, will have to work with computers. If they only plan on taking one course, then they should be presented with all the basics. I would fuse together 1001 with a little 1005 and toss in some hardware work (ie. what a motherboard is, what the CPU does, the different buses, and the add in cards. Plus a basic explanation of the various ports, like IEEE1284, 1394, RS232, USB, and so forth. They will encounter these devices most everyday, and should not falter when one of these devices is presented to them.

126 ECE 2004 Good theory coverage. Problems relevant to the major. Good and understanding instructors.

127 ECE 2004 Graphics.

129 ECE 2006 isnt cs1005 an introductory course?

130 ECE 2006 Yes, I would like to have some knowledge of programming.

131 ECE 2006 Basics on the languages and basic ideas and concepts behind what things do.

132 ECE 2005 I do not know that much about what is currently offered, but an intro to object oriented programming, while touching on assembler seems reasonable. The ability to write programs to perform mathematical functions or simple text manipulation is very valuable in an intro course. I think an intro course, one should do programming in a text editor, such as Emacs, NOT in an independent development environment such as codewarrior. These programs shelter the user from understanding how to really put a program in place without the IDE.

133 ECE 2005 As much theory as possible.

134 ECE 2005 Use Basic... Best way to teach basic concepts without worying about more complex issues. VB is also a likely laguage to be used in the field.

135 ECE 2005 Purhaps

136 ECE 2005 I think a CS course should be created for non-CS majors, which include software programming with some hardware introduction.

137 ECE TR all the basics of programming

138 ECE 2005 more help

139 ECE 2005 Yes.

140 ECE 2005 HTML

141 ECE 2005 Something that focuses or is applicable to the particular non-CS major.

142 ECE 2005 teach how to program

143 ECE 2005 Adequate Coverage of Algorithms before any actual programming .

144 ECE 2005 Go slowley, helps the non-cs majors.

145 ECE 2005 It would be useful to all non-CS majors as well as CS majors for the topic of Matlab and how programming is related to it be touched on. Possibly some exercises... not too extensive though. Just enought to be familier with the syntax.

147 ECE 2005 the basics i guess like the format off programming what it is and basic loops

149 ECE 2005 Major related programming

150 ECE 2005 The same general material, but easier (take less time to complete) homeworks, and spend a little more time on each topic for those of us who have never heard of a lot of the stuff being taught to us.

152 ECE 2005 C++/Java

153 ECE 2002 Would like the proffesor to start on the assumtion that the topic has never been introduced before to the student

154 ECE 2005 More of the thinking process of programming. Like algorithms, and thinking of ways how you approach problems.

155 ECE 2005 C++ and HTML

156 ECE 2005 i think that it was taught very well

157 ECE 2005 programming microprocessors such as the PIC, assembly, in my opinion was the easyest programming language to learn

158 ECE 2005 Something easy

159 ECE 2005 i wouldnt change a thing

160 ECE 2005 Matlab, C++ and less conceptual

161 ECE 2002 C programming, covered pratically, not just theoetically

162 ECE/HU 2005 more examples

163 CS 2006 Lot's of hands-on programing with minimum theory and maximum practical applications

164 CS 2006 I'd like to see a little more generalized programming. When I was in my second year of 'CS' (more ECE than CS, really) in high school we spent a little time programming. We went through and learned how to program similar simple programs in a few of the easier languages. (Assembly, BASIC, C)

165 CS 2005 Perhaps a little bit more of the basics before jumping right in, i.e. covering data types, what a funciton is, etc.

166 CS 2005 I think the introductory course for CS majors is suitable for non-CS majors.

167 CS 2006 Basic programming skills up to object oriented programing.

169 CS 200? Definitly, I think this is a great idea.

170 CS 2006 A little more basics

171 CS 2005 1005, 2005

172 CS 2005 More real world examples of how it applies to their major, and how it can be useful to them.

174 CS 2006 Basic skills, less theory. Popular language, general topics.

175 CS 2005 start with basic c++, then move on to show the robustness oft the language.

176 CS 2003 Slower pace. Use more practical languages like C/C++ and Java as opposed to Fortran

177 CS 2005 Projects that show a relation to their majors, not just programming random stuff.

178 CS 2005 Yes, it would give everyone a decent background in the subject area

179 CS 2005 dont know didn't take the class

180 CS 2003 No

183 CS 2006 Practical grounding in one of the general-purpose languages.

184 CS 2005 More concepts, less programming. After all, they are not going to need to program, but know the concepts are important if and when they work with programmers

185 CS 2005 easy java/ or scheme(because of built in lists and not using

186 CS 2005 Split them up into groups by major and expose them to ways that programming could possibly affect their major, or ways they could use that skill to help themselves in their major.

187 CS 2005 everything in cs1005

188 CS 2005 i think an introductory course would be a good idea, many people will need to know how to read code when they get jobs. and if they don't need it in their job, it may be helpful elsewhere

189 CS 2005 A slower toned down approach to not only learning how to program in a certain language but how to program in general.

190 CS 2005 More emphasis on programming theory than practical usage

191 CS 2005 Basic concepts and programming skills like data types and basic techniques

192 CS 2005 More what programming is and not dive right into the language.

194 CS 2005 A lot of syntax being covered.

195 CS 2004 If there was a course I think that it should introduce students to general programming techniques, etc. Some students find it very difficult to learn programming in C, C++ if they have no interest in it. I think the courses should be alittle easier for non-CS majors.

198 CS/EE 2005 universal syntax generalities, what compilers are, how to find info on programming specifics, basics of computer architecture, software, and hardware.

199 CS/EE 2004 yes

18) General Comments:

12 BC 2003 Taking an Intro to C course without knowing what to expect, it would be nice to have a survey or somesort ot make sure you understand the prereqs for the course before its too late.

16 BE 2003 Non-CS majors do not have a lower ability to program at a basic level. They don't need a special course to teach them.

21 CE 2004 If you do not have an interest in programming it can be difficult, boring, and time consuming.

26 CE 2005 It would help to have a CS course that anyone can take and understand.

52 MAC 2006 Having not taken any courses here yet, I cannot give much input. I have heard good comments from my friends who have taken the courses, but I don't know any specific criticisms.

60 ME 2005 most WPI students have enough general computer knowlage to handle basic programing situations without a required course

74 ME 2005 I dont really know that much about programming.. Learning about it would be interesting

75 ME 2005 I've never taken any courses. I was kinda interested at first, but after all the horror stories I've heard? I might still take one then again I might not.

76 ME 2003 Computer programing is a useful skill it also helps promote anylitical thinking.

78 ME 2005 none

79 ME 2004 WPI Intro to CS needs concrete examples, not contrived abstracts.

82 MFE 2006 Regarding the language below... / I barely understand any of it, which is why I am taking CS 1005 in the first place. However, language B seems to be a bit more simplistic, but I may be completely mistaken.

83 MFE CS at WPI is a pretty hard

84 MFE 2005 I'm not interested in CS at all, only taking this survey for a professor, so I'm sorry to give you nothing useful.

85 MGE 2003 The CS program for non majors really needs to get started or get better or something-although CS is one of the three biggest majors at wpi, there are atkeast 50 percent of the wpi community who isnt a CS major and NO non cs major courses offered...thats really not appropriate.

86 MIS 2004 I only took CS 2005 because it was required for my major at the time. It wasn't horrible but I probably wouldn't have taken it if I'd known it was going to be dropped as a requirement for MIS majors. I'll never use the stuff again.

89 ND 2006 A course that would teach the basics would be very helpful.

101 TC 2003 I've heard that the intro CS classes (ie CS 1005) are much too hard if you come to WPI to learn programming and haven't learned it before.

102 EE MS If by the last question, you are implying that a generic language might be created for WPI CS... MIT has already done this with a language called CLU (pronounced 'CLUE'. / My brother and sister in law went to MIT and learned in CLU and she is now working for Microsoft, and he is in the computer gaming industry.) / http://www.pmg.lcs.mit.edu/CLU.html / On the other hand, I'd teach them what is commonly in use. Sliderules are cool, but... / / Also, I'd teach an infix notational language because it is more like math and therefore the thought processes learned are probably more generally applicable (only, I'd pick a standard one.)

103 EE 2006 Good luck with your IQP.

108 EE 2003 the theory behind the programming languages should perhaps be left to CS majors to worry about. although a basic understanding of how and why a structure is being used is necessary, i don't see the point of too much depth being chosen over sufficient breadth in the introductory courses.

109 EE 2005 CS intro classes should be smaller and not taught out of the book, more examples and less theory.

119 EE 2003 Aesthetically, I prefer 'high-level' languages like Matlab, Basic, VHDL to 'low-level' languages like assembly. Sure, low-level languages may be a more efficient use of hardware in some cases, but writing code is more labor intensive, and therefore a less efficient use of human resources. / / I especially like Hardware Description Languages (HDL) because they are 'high-level' (behavioral), efficient, very-fast, powerful, and embeddable, and they utilize familiar EE knowledge (we already think in terms of hardware). From the point of view of an EE student, I consider program counters, and pointers to be abstract concepts.

Software always runs on hardware anyway, so EEs can and should use their knowledge to their advantage when programming.

125 ECE 2005 I like the idea of a programming course for non CS folks, and I believe the current 1000 level courses offer what is needed. However since most will only take one course, I think it is important to design a special class JUST for non CS/ECE majors. I outlined an idea above.

126 ECE 2004 Most of the times if a non-CS major is taking for example CS 2005, they are at a huge disadvantage. They need different level of learning and the instructor spends most of the time discussing advanced theories with the students who already know the material. / I personally think knowing how to program is important, but requiring a generic course of everyone is not a good approach.

132 ECE 2005 My understanding is that Java isn't particularly taught at WPI. My intro course (at Harvard Extension School) was in Java, and I wonder that it wouldn't be valuable to teach. I have also heard that 2005 is fairly easy, which is too bad since Data Structures is an interesting and challengin subject.

135 ECE 2005 I knew a little bit about programing before I came here and it helped but the course was really good.

138 ECE 2005 I just don't believe that the teachers take into account that not everyone has been programming since they were 10 years old.

141 ECE 2005 I am currently taking CS2005. Many ECE majors that I have talked to have failed the course at least once, most only achieving a C the second time around. I feel this is too excessive for non-CS majors, since it takes their concentration off of courses related to their major.

142 ECE 2005 programming is something that some people can do naturally and other are bio majors

143 ECE 2005 I took CS 2005, as I had taken AP computer Science in high school, and got credit for cs 1005 for it. I was very surprised by cs 2005, as it was completely different in regards to expectations and how things were graded, although this is not to say that the course was bad. It was simply different. /

145 ECE 2005 When I interned last summer at Bose Corp., the engineers there almost assumed knowledge of Matlab. All the engineers there used it... from EE to ME etc...

150 ECE 2005 CS1005, for an INTRODUCTORY course, at least the one I took (A term, 2002, Professor Joe Wong's class) is far too hard. I'm having an easier time in CS2005 than I did in 1005. But I heard I would need that class for 2005, which is the easiest way to fulfill the CS requirement for ECE majors. / An introductory course that focuses on basics rather than dabbling a short time in lots of topics would be far better. / Question 19: Both these languages stink: C++ is easy to learn, stick with that.

155 ECE 2005 WPI offers many very good classes on programming. I wish they were more a part of the ECE program.

157 ECE 2005 cs is too hard in most of the classes / / ABOVE ALL ELSE: / SCHEME SUCKS

158 ECE 2005 I don't think programming should be required unless you are a CS major.

164 CS 2006 The C++ course wasn't really worthwile to me because I'd just gotten out of a year of C programming and the only difference between C and C++ are the commands and the object oriented stuff, but we didn't even get into the OO stuff until the last week. It wasn't included on anything but one lab, and the way the grading system was set up if you'd gone to enough of the other Wednesday labs/Tuesday classes you didn't need to show up for it.

165 CS 2005 The above questions were answered with regards to CS 1006 and do not reflect my opinions on the other courses listed.

169 CS 200? I really hope that the a course can be designed for all majors to teach practical programing. / / Definitly Language B!!

186 CS 2005 As I'm a CS major filling this out for 2 homework assignment bonuses, you should probably disregard this survey. But I did try to fill it out honestly and fully, so take it as you will. / Also, making non-CS majors make a choice between prefix programming and infix programming is a real bitch move, because the prefix looks ugly, and they won't have used it, so of course they'll pick infix. That's a really biased question, and I don't think the results of it are indicative of anything except that people who don't know much about CS like easy-looking languages.

189 CS 2005 I feel that only a lower introductory programming class should be required of non-CS majors as some of the classes (i.e. CS2005) are too much for someone with that little programming experience to be expected to pass.

199 CS/EE 2004 this survey is poorly desinged. Since I am able to select MULTIPLE choices for both CS courses and many other options, it is not clear that you are interested

in a particular CS class. AS you can see from my response up top, I have taken MANY cs classes and they were all on a different level. You are not asking me to evaluate an introductory class specifically, but rather evaluate _ALL_ my CS classes at one time, which is IMPOSSIBLE.

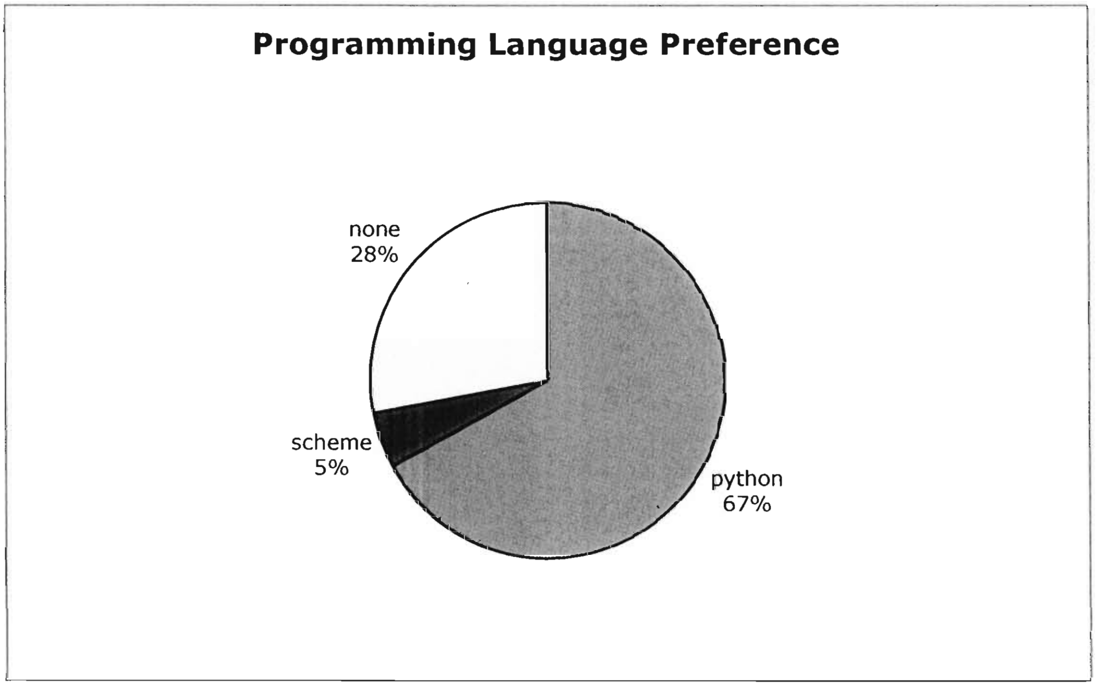
IV. Language Preference

19) Below are two languages we are considering for an introductory computer science course. If you prefer one over the other, please mark below:

language A (Scheme)	language B (Python)

;; factorial
(define (fact x)
  (if (<= x 1) 1
     (+ x (fact (- x 1)))))
	# factorial
	def fact(x):
	   if (x <= 1):
	      return 1
	   else:
	      return fact(x-1) * x

;; quadratic equation
(define (quadeq a b c)
  (let [(disc
        (sqrt (- (expt b 2) (* 4 a c))))]
    (cons (/ (+ (- b) disc) (* 2 a))
          (cons (/ (- (- b) disc) (* 2 a))
                '()))))
	# quadratic equation
	from cmath import sqrt
	def quadeq(a, b, c):
	   disc = sqrt(b**2 - 4*a*c)
	   ans1 = (-b + disc)/2*a
	   ans2 = (-b - disc)/2*a
	
	   return (ans1, ans2)



Programming Language Preference

A.2 Faculty Interviews and Surveys

A.2.1 Interview analysis

After we issued a survey to students, we sought feedback from faculty of different departments. We started by conducting individual interviews. P.K. Aravind, Terri Camesano, John Goulet, Judith Miller and Creighton Peet were gracious enough to let us interview them. We asked about the current introductory CS sequence as well as what they would like to see their students learn in an introductory CS course.

We received a range of opinions about the current introductory CS courses. One professor thought the current introductory CS courses did the job. Students who did projects with this professor use programming extensively in MQPs and are fluent in C/C++/Java. Another professor expressed the opposite view about the current introductory course: "The students who have taken [CS 1001] feel it's pretty useless".

When we asked about what faculty would want to see in an introductory course the responses were similar. Most mentioned knowledge of algorithms, looping, and logic. A few professors noted that knowledge of MatLab or Maple would be very useful since students use those packages in other classes.

During our interviews with the faculty, a few exceptional ideas were hit upon. Professor Goulet suggested that the last week of an introductory programming course be dedicated to a particular software package which students will use in later courses. For math majors the last week would be dedicated to learning Maple, for chemical engineering the last week would be dedicated to learning MathCAD, etc. We feel this would be an excellent idea, but would not fit very well in the WPI's condensed seven-week term. This idea would be better suited at a semester based school. Professor Camesano also brought an important point to our attention. The schedule of a chemical engineering student doesn't allow time for a CS course, but it does allow time for engineering electives. This is the reason we came to the conclusion that our course would be more accessible if it were cross-listed as a computer science and engineering science course.

A.2.2 Survey analysis

Following our interviews with the faculty we got a clear picture about what the faculty wanted in an introductory CS course. We used the data from the faculty interviews to create a survey. We asked the faculty to rank particular skills and concepts in order of importance as well as some open ended questions. The purpose of ranking the skills and concepts gave us input for creating a course syllabus. The open ended questions were to confirm our own ideas and to address problems with the new CS proposal.

We received a total of seven responses from six different departments. The ranking section of our survey was not as effective as we had hoped. Some respondents didn't feel comfortable answering many of those questions because they felt they didn't have sufficient knowledge about the idea or concept. We anticipated this so we created a glossary where a respondent would click on the word to read a description about the idea or concept. The feedback we received on that section of the survey indicated our glossary was not informative enough.

For the open-ended section of the survey we wanted to reinforce some of our ideas about our recommendations as well as the WPI course proposal. We hypothesized that listing the course as Engineering Science would make the course more accessible to students. Also we hypothesized that separate courses for majors and non-majors would be beneficial, as opposed to the WPI proposal where they suggest one course for all majors. Finally we hypothesized that non-majors will only be expected to take one course instead of the two courses which the WPI proposal suggests.

The feedback regarding the possibility of having the course be Engineering Science was positive, overall. Two responses indicated an ES course would be better for the students and the rest of the responses didn't know if that would help.

The response regarding separate courses for majors and non-majors reinforced our hypotheses. Most of the responses indicated that intimidation would be a factor for students. Responses also indicated that the two groups, majors and non-majors, have different needs and goals for taking a CS course.

> *I've also spoken with non-CS majors who would like to take a programming course, but not with CS majors. They feel that if they took a course with intense CS majors, they would be left behind and they wouldn't end up learning much. It seems to me that their needs may be very different.*

Our last hypotheses concerned the number of CS courses a non-major should expect to take. The majority of the responses we received said that non-majors would be expected to take only one course in CS.

A.2.3 Faculty survey questions

```
Background on the project

  This survey is part of an IQP to create a course to teach non-CS
  majors about programming in a practical environment. More information
  can be found on our [1]project page.

  Our goal is to find out what you feel is important for your students
  to learn in an introductory CS class. We are attempting to compile a
```

list of skills that your students will likely use in other courses at WPI and their future careers, so our course can be as beneficial to your students as possible.

Part 0: Your information

Please fill out the information below; this information will be kept confidential.

Name ____________________
Email ____________________
Department ___

May we contact you for more information regarding this survey?
Yes (_) / No (_)

Part 1: Rank Skills

Please rate each of the following concepts as it applies to your students on a scale of 1-5, where

1 Very unimportant
2 Unimportant
3 Neutral
4 Important
5 Very Important
NA Not Applicable

	NA	1	2	3	4	5
- Programming -						
- Arrays and lists	(_)	(_)	(_)	(_)	(_)	(_)
- Knowledge of a specific programming language	(_)	(_)	(_)	(_)	(_)	(_)
- Using a language with syntax similar to C/Maple/MatLab	(_)	(_)	(_)	(_)	(_)	(_)
- Object Oriented Programming	(_)	(_)	(_)	(_)	(_)	(_)
- Recursion	(_)	(_)	(_)	(_)	(_)	(_)
- File input/output	(_)	(_)	(_)	(_)	(_)	(_)
- Looping	(_)	(_)	(_)	(_)	(_)	(_)
- Implementing data structures	(_)	(_)	(_)	(_)	(_)	(_)
- String manipulation	(_)	(_)	(_)	(_)	(_)	(_)
- Modular code and functions	(_)	(_)	(_)	(_)	(_)	(_)
- Algorithms	(_)	(_)	(_)	(_)	(_)	(_)
- Understanding syntax and errors	(_)	(_)	(_)	(_)	(_)	(_)
- Major-specific projects	(_)	(_)	(_)	(_)	(_)	(_)
- Design -						
- General problem-solving skills	(_)	(_)	(_)	(_)	(_)	(_)
- Knowing where to look for resources and help	(_)	(_)	(_)	(_)	(_)	(_)
- Breaking up a problem	(_)	(_)	(_)	(_)	(_)	(_)
- Program design and design recipes	(_)	(_)	(_)	(_)	(_)	(_)
- Logical thinking skills	(_)	(_)	(_)	(_)	(_)	(_)
- Debugging	(_)	(_)	(_)	(_)	(_)	(_)
- Flow charts	(_)	(_)	(_)	(_)	(_)	(_)

- Mathematics -

- Computer precision and rounding error	(_)	(_)	(_)	(_)	(_)	(_)
- Using math functions	(_)	(_)	(_)	(_)	(_)	(_)
- Random numbers	(_)	(_)	(_)	(_)	(_)	(_)
- Complex numbers	(_)	(_)	(_)	(_)	(_)	(_)
- Matrices	(_)	(_)	(_)	(_)	(_)	(_)
- Ability to translate math/science equations/concepts to code	(_)	(_)	(_)	(_)	(_)	(_)
- Boolean algebra	(_)	(_)	(_)	(_)	(_)	(_)
- Applications and Advanced topics	(_)	(_)	(_)	(_)	(_)	(_)
- Graphing and plotting data	(_)	(_)	(_)	(_)	(_)	(_)
- Web programming and CGI	(_)	(_)	(_)	(_)	(_)	(_)
- Graphics programming	(_)	(_)	(_)	(_)	(_)	(_)
- GUI programming	(_)	(_)	(_)	(_)	(_)	(_)
- Pattern matching and regular expressions	(_)	(_)	(_)	(_)	(_)	(_)

Comments and/or additions to choices above:

Part 2: Open ended questions

Please answer each of the following questions.

2) Do you feel that a CS class with a 1000-level designation would be better received
than one with a 2000-level designation (by non-majors)? Why?

----

3) Are you familiar with the CS departments' new introductory CS curriculum, including a combined introductory sequence for all students? ([19]Available in PDF form here)
If so, do you believe this proposal will meet your students' computer science/programming needs? Why or why not?

----

4) Would an introductory programming course fit better into your students' schedules as a CS or ES (Engineering Science) course? Why?

----

5) Do you feel that creating separate introductory CS classes for majors and non-majors would create a better experience for non-CS

majors? Why or why not?

----

6) How many CS classes is a reasonable expectation for a major in your field to take? Please elaborate if necessary.

(_) NA (_) 0 (_) 1 (_) 2 (_) 3 or more

----

7) Do you have any ideas for sample programming projects which would involve your area of study, but are simple enough for a first-year student to understand?
Also, would you be willing to be contacted to give more information about such an assignment?

----

8) Other comments:

A.2.4 Faculty survey results

0 NA
1 Very unimportant
2 Unimportant
3 Neutral
4 Important
5 Very Important

Programming							
Arrays and lists	2		1	1	4	0	5
Knowledge of a specific programming language	3		1	2	4	0	3
Using a language with syntax similar to C/Maple/MatLab	5	5	1	1	3	0	4
Object Oriented Programming	2		1	1	5	0	4
Recursion	3		1	1	5	0	4
File input/output	4		1	1	5	0	5
Looping	3		1	1	4	0	5
Implementing data structures	3		1	2	5	0	4
String manipulation	3		1	2	4	0	3
Modular code and functions	3		1	1	4	0	3
Algorithms	4		3	2	5	0	4
Understanding syntax and errors	4		3	2	5	0	4
Major-specific projects	5		3	2	4	0	3
Design							
General problem-solving skills	5		5	5	5	5	3
Knowing where to look for resources and help	5		5	5	5	4	3
Breaking up a problem	5		3	5	4	5	3
Program design and design recipes	3		2	3	5	4	3
Logical thinking skills	5		5	5	5	5	3
Debugging	5		3	3	3	4	5
Flow charts	5		2	4	4	4	4
Mathematics							
Computer precision and rounding error	4		4	2	3	0	3
Using math functions	5	5	4	4	3	0	5
Random numbers	1		2	4	3	0	3
Complex numbers	2		1	1	4	0	4
Matrices	4		1	2	4	0	5
Ability to translate math/science equations/concepts to code	5	5	4	2	4	0	5
Boolean algebra	2		3	5	4	0	2
Applications and Advanced topics							
Graphing and plotting data	5	5	5	5	4	0	4
Web programming and CGI	4	5	4	1	3	0	2
Graphics programming	3		3	1	5	0	2
GUI programming	3		2	1	3	0	2
Pattern matching							

and regular expressions	1	1	2	4	0	1
RComments			My strange pattern of responses has to do with the importance of using Excel in graphing data in my class. They have to be able to handle functions but not code them on their own.		The reason I answered "NA" to all but the Design section is because honestly I don't feel sufficiently informed about the issues to be able to judge the importance of those specific skills to students these days. The skills/abilities in the Design section seemed sufficiently broad that they would contribute well to the education of any WPI student-- that's my instinct. Even if there is "overlap" with design skills taught in my discipline, I think it would be good for students to see it *applied* to computer program design. I almost feel as if recent non-CS alumni would know better how to weigh the importance of some of the other areas.	
2)Level	maybe	yes, less intimidating	Sure -- it would seem less involved.		Yes. I think it would be less intimidating, especially for students who have no prior CS or programming experience from high school.	Yes. It implies that previous CS knowledge is not required.
3)New initiatve	CS courses don't satisfy any specific CM requirement, so my guess is that relatively few students take CS courses. Only students with a specific interest or who plan to go to grad school is some other field end up taking CS.	yes, I would recommend that they take the first course only. It doesn't matter to me what language they learn, it's the thinking skills that are importnat.	Not that familiar - I'm sure it will meet their minimal programming needs in my class.	don't know	For some students, perhaps, but I've also spoken with non-CS majors who would like to take a programming course, but not with CS majors (no offense). They feel that if they took a course with intense CS majors, they would be left behind and they wouldn't end up learning much. It seems to me that their needs may be very different.	
4)CS vs ES	Students in my department would be better off taking a course listed as "ES" since they could count this as an engineering elective. They must take 3 engineering electives. It is not that they wouldn't benefit from the course or that they are not interested, but having the course crosslisted as ES or EE would help students meet certain	doesn't matter	I don't know.	don't know	I don't think it matters for ME majors, but to be honest I'm not really sure. I think that either way, the course would be "counted" toward our overall "engineering science and design" requirement, but I'm not 100% sure! The person to ask in ME about whether CS courses "count" in our general engineering science and design category is Prof. Zhikun Hou, who is the one who currently checks audits.	ES: a non-CS is interested in programming skills to address engineering situations with a numerical solution.

	requirements.						
5)Separate majors from non majors	probably. Non-CS majors might feel intimidated if CS majors already have a lot of background programming knowledge and the non-majors have no prior background.		I think it might create a better experience for CS majors, because it would be a better balanced and more diverse group of students.	Perhaps -- the expectation of prior training would be lower for a non-majors class.	possibly	Yes, I think there would be less variation in prior programming experience, which would make the non-majors feel "less stupid" and in a lower-pressure environment.	Yes: CS majors have numerous other interests associated with the science of CS. non-CS majors are interested in the engineering (not science) aspects.
6a)# of classes			1	1	2	1	2
6b)# of classes comment	0 or 1			As people get more training in psychology, it is helpful to program displays and data collection for experimental studies on computers. Our students don't currently require that level of advanced knowledge.			
7)sample projects	maybe something in reaction engineering, but this is not my area of expertise.		Personally, no.	Sure -- they could program a sequence of displays for people to respond to and record responses and reaction times.		Hmm, not really. An alternative to trying to tie it to particular areas of study would be to try to select projects that almost any student would relate to from their everyday lives/ activities. In other words, try to engage their interest from the co-curricular realm rather than in their major discipline? Unfortunately I can't think of any specific ideas, but it's a possible alternative strategy that wouldn't require you to think of "n" different programming projects for "n" different majors.	Yes to both.
		My research students sometimes need to write their own programs for instrument control and data manipulation. We have used Excel, Sigma Plot, MatLab, Maple, and HPVEE. Once one becomes familiar with one kind of programming, it's easy to transfer that thinking to other programs, so even if students don't get instruction in the specific programs that I mention, just learning about how computers work is valuable. The graduates of a tech school should all know how					

8)Other Comments	- surf the Internet, download information from a site, use a spreadsheet, take a quiz on the computer, use a word processor and presentation software, and write a web page. I agree that it would be good to have a CS course for non-CS majors that teaches students to be competent computer users. For the computer-skittish, the gentlest and most fun way to introduce them to actual programming is web-page design, I think. I believe this is a requirement for undergraduates at Dartmouth.		contact.

Appendix B

Transcript of Interview with Elkner

Wed Mar 19 15:00:02 2003

Brad Noyes: Welcome everyone. Thanks for making it. We are interviewing Jeff who is in VA. Brad and Brian are in a lab at WPI, and Prof Carolann Koleci is in her office, at WPI. So let's begin.

Jeff Elkner: *hi everyone*

Carolann Koleci: Hi all!

Brian Corcoran: Just to remind Jeff, our project is on creating a Programmining Course for non-CS-majors, using Python.

Jeff Elkner: *sounds good*

Jeff Elkner: *my only direct experience is with high school students*

Carolann Koleci: Question regarding area of concentration, is there a CS teach exam in VA high schools?

Jeff Elkner: *no*

Carolann Koleci: What is the size of the class and the size of the school (student population wise)?

Jeff Elkner: *school: 1500*

Jeff Elkner: *class: 2 sections of 25 each*

Brian Corcoran: Are there other CS classes offered besides the one you teach?

Jeff Elkner: *i was only describing CS I and II, our 1st year class*

Jeff Elkner: *we also have AP CSC, Network Operations (CISCO), and CIS Advanced Topics*

Brad Noyes: so in total there are 5 CS classes there?

Jeff Elkner: *yes*

Brad Noyes: Are these classes each half year?

Jeff Elkner: *no, they are all full year, execpt CS I and II, which are two half years taught back to back*

Jeff Elkner: *making them effectively a full year*

Carolann Koleci: Do you use any other standard way of measuring your student's abilities? In physics we national assessment tools that we use to measure conceptual gain of understanding, anything similar for CS?

Jeff Elkner: *unfortunately, i don't have any effective formal measurement tools at present*

Jeff Elkner: *CS classes are electives, and VA has not put any standard tests in place*

Brad Noyes: What else have you taught at the HS level?

Jeff Elkner: *i've been teaching for 12 years, math and cs*

Carolann Koleci: Common problems the you see in the student's learning, is there a particular area where student's have trouble?

Jeff Elkner: *there are several*

Carolann Koleci: what are some of the most prominent ones?

Jeff Elkner: *the biggest one is getting them to think algorithmically*

Carolann Koleci: We see that in physics too.

Brian Corcoran: what exactly do you mean by that?

Jeff Elkner: *to understand what a computer is capable of doing and being able to express what they want it to do in terms the machine can understand*

Jeff Elkner: *it is also what i like most about teaching this subject*

Brian Corcoran: Our other co-advisor, who could not make it, saw a general problem with abstraction,

Jeff Elkner: *i tell my students throughout the year, "Computers are dumb as rocks! It is up to you to make them do clever things"*

Jeff Elkner: *yes, abstraction is difficult*

Brian Corcoran: in particular, functions. Have you experienced this?

Jeff Elkner: *yes, and the nice thing about doing this in a CS class is i get to reinforce what they are learning in math*

Jeff Elkner: *in a way that is more fun for many students*

Jeff Elkner: *writing functions is what we do all year*

Brian Corcoran: That idea is a basic part of the class we hope to create; using programming and science/math to reinforce one another.

Jeff Elkner: *it is a wonderful idea*

Jeff Elkner: *and it will work very well, because they naturally reinforce each other*

Brian Corcoran: Do you find that separating funcitions and "fruitful functions" help students understand the concept better?

Jeff Elkner: *i tend to focus on a functional style of programming*

Carolann Koleci: How many students, or what percentage, are able to make the transition to think algorithmically, and what do you feel is most responsible for this?

Jeff Elkner: *i see a definite progress throughout the year with most students achieving at least a basic concept of what an algorithm is and how to write one.*

Brian Corcoran: You mentioned that you focus on a functional style of programming... Did you consider Lisp or Scheme for your class? If so, why did you chose Python over them?

Jeff Elkner: *i did look at the teach scheme project*

Jeff Elkner: *but i think Python's syntax is so much easier to handle.*

Brian Corcoran: That's our view as well.

Carolann Koleci: What kind of applications do you use/provide?

Jeff Elkner: *besides, Python provides the flexibility to look at multiple approaches to programming*

Jeff Elkner: *not just functional*

Jeff Elkner: *not just OO*

Brian Corcoran: Do you cover OO programing in your class?

Brian Corcoran: and how much?

Jeff Elkner: *yes*

Jeff Elkner: *we will be starting that the last quarter*

Jeff Elkner: *we will develop a Card and Deck classes and look at some simple card games*

Brian Corcoran: do you do any graphical programming?

Jeff Elkner: *yes and no*

Jeff Elkner: *students do independent projects at the end of the year*

Jeff Elkner: *many of them choose to look at TkInter, wxPython, anygui*

Carolann Koleci: so your students have the option to do graph. programming?

Jeff Elkner: *this year two of my students are looking at VPython*

Jeff Elkner: *yes*

Brad Noyes: We've looked at VPython as well, what are your students doing with it?

Jeff Elkner: *in fact, pyKarel was developed by two of my students (see [1]http://pykarel.sf.net)*

Brad Noyes: oh okay.

Brian Corcoran: What other packages or programs to you use in your class?

Brian Corcoran: For example, IDLE

Jeff Elkner: *i have also used the livewires python course materials*

Jeff Elkner: *we are using nedit as our editor and running programs from the command prompt*

Brian Corcoran: have you considering using an IDE?

Jeff Elkner: *i tried using IDLE earlier in the year, but the key bindings conflicted with our GUI*

Jeff Elkner: *we have too many problems*

Jeff Elkner: *besides, Python provides everything you need write in the shell*

Carolann Koleci: Do you know of Other colleges/HS that use python?

Jeff Elkner: *yes*

Carolann Koleci: Which colleges?

Jeff Elkner: *hold on, let me get you a url*

Carolann Koleci: thanks!

Jeff Elkner: *[2]http://www.ibiblio.org/obp/pyBiblio/schools.php*

Jeff Elkner: *i'm hoping to add your school to this list ;-)*

Carolann Koleci: :-)

Brian Corcoran: We hope to as well

Brad Noyes: Can you list some of the pros and cons of switching to python?

Jeff Elkner: *the biggest pro has been that the simplicity and clarity of Python syntax has enabled me to do so much more in so much less time*

Jeff Elkner: *the biggest con was all the work it entailed for me, since there wasn't any teaching materials available*

Jeff Elkner: *that is changing now*

Brad Noyes: Ahh, are there any cons for the students?

Jeff Elkner: *if the goal is to teach students abstract thinking skills and give them a taste of what computer programming is, then i would say no*

Jeff Elkner: *Python is *so* much better than C++ for those goals*

Brian Corcoran: How do students find the transition from your class to the AP CS course?

Jeff Elkner: *only a small percentage of the students go on to AP, and their reactions vary.*

Jeff Elkner: *most are ready by then for a language that requires more thinking at a level closer to the machine hardware, so they have fun with C++*

Jeff Elkner: *next year it will be Java*

Brian Corcoran: How large is the AP CS class?

Jeff Elkner: *This year I only have 4 students*

Jeff Elkner: *next year it will probably be more like 15*

Carolann Koleci: Have you noticed gender bariers in your classes? Do fewer female students continue on to the other CS class?

Jeff Elkner: *well my numbers are so small it is hard to reach meaningful conclusions*

Jeff Elkner: *1 of the 4 AP students is female*

Jeff Elkner: *which is a higher percentage then the intro classes*

Jeff Elkner: *but if she wasn't there it would be 0*

Brian Corcoran: What percentage of your intro class is female?

Jeff Elkner: *give me a minute*

Jeff Elkner: *5 of 25 in 1st period*

Jeff Elkner: *6 of 24 in 2nd period*

Brian Corcoran: In general, how do female students rank in the class? Do you find that they respond to particular methods/projects different than the male students?

Jeff Elkner: *the female students tend to be above average.*

Carolann Koleci: Any idea if this correlates with math ability?

Jeff Elkner: *i always figure that they had the independence of spirit to buck the trend by signing up for the class in the first place*

Jeff Elkner: *i've seen a lot of discussion about whether different approaches to teaching programming would be more effective with female students, but i can't say i've found anything conclusive*

Brad Noyes: Since you have switched to python, has the enrollment been larger in your class then in the past?

Jeff Elkner: *yes it has, but i don't necessarily think that has to do with python*

Jeff Elkner: *our school has been shifting demographically*

Jeff Elkner: *i'm finding that female students from other countires are more likely to take CS classes*

Carolann Koleci: interesting

Brad Noyes: What about enrollment in general for your class?

Jeff Elkner: *it is about twice what is was before we started using python*

Jeff Elkner: *but again, that has a lot to do with what i want to do with the class*

Brian Corcoran: what do you mean?

Jeff Elkner: *i want it to be more of a liberal arts class than a stepping stone only for future engineers and CS majors*

Carolann Koleci: Has the level of enthusiasm in students changed since you started to teach python?

Jeff Elkner: *yes, but the best way to really measure that is at the top end*

Carolann Koleci: top end?

Jeff Elkner: *i would say that most of my students still leave the class thinking that programming was interesting, but not something they would want to do in their spare time*

Jeff Elkner: *for the kids who want to program, Python really stands out*

Brian Corcoran: Do any students use python or programming in other classes, or outside of the classroom?

Jeff Elkner: *because they can reach their goal of writing real, powerful applications in much shorter time.*

Jeff Elkner: *outside the classroom, certainly*

Brian Corcoran: Have you done any collaboration with other departments?

Jeff Elkner: *there is always a small group of students who spend more time than they should programming*

Jeff Elkner: *yes, i have a student now working on a program with a biology teacher*

Brian Corcoran: Can you elaborate on that?

Jeff Elkner: *he is working on a program that graphs data that students are gathering in the biology class*

Jeff Elkner: *last year 2 students did a science fair project similating reproduction of fruit flies*

Brian Corcoran: Can we get more information about these projects, after the interview?

Jeff Elkner: *should brad be the email i use to contact you?*

Brian Corcoran: That's fine.

Brad Noyes: We have noticed that students now start to develop their own projects (pyKarel, Zuite). Did students do this before you started to teach python?

Jeff Elkner: *not to anywhere near the degree they do now.*

Jeff Elkner: *this is what i meant by "the top end"*

Jeff Elkner: *with Python programming is faster to learn and easier to be productive*

Jeff Elkner: *students are now capable of doing things they could not have done before*

Jeff Elkner: *three students this year are working on pyJotto, and program called SpellQuest*

Jeff Elkner: *in each of these projects i have been able to hook up interested students with professional programmers who mentor them*

Carolann Koleci: that sounds wonderful

Brian Corcoran: Do these professionals usually know python?

Jeff Elkner: *yes*

Jeff Elkner: *i find them in the Python community ;-)*

Jeff Elkner: *i look for a professional programmer who already has a cool project they are working on in their spare time.*

Jeff Elkner: *i offer them the following win-win deal: you mentor my kids, and they will help you with your program.*

Carolann Koleci: Do these projects turn into summer internships?

Jeff Elkner: *we now have a relationship with a local software company that has hired our top student for the past two years.*

Jeff Elkner: *i mean one student last year, and one this year.*

Brad Noyes: In your email you mentioned having some students participate with us. Since our idea is to integrate science and programming we would like ask if your students would be able to do some projects that we come up with about science.

Brad Noyes: Our goal would be to have your students do the projects and see what they learn about the area of science they project was based on.

Jeff Elkner: *that would be great!*

Brad Noyes: excellent!

Jeff Elkner: *what science areas do you have in mind?*

Brad Noyes: physics, Bio ... perhaps Chem and math.

Jeff Elkner: *i have two students who have expressed an interest in doing something with chemistry*

Carolann Koleci: inorganic, organic, preferences?

Jeff Elkner: *keep in mind that we are a high school, i don't know how much organic they do.*

Carolann Koleci: Ah, very true.

Jeff Elkner: *that is the biggest potential problem, we are at different levels of academia*

Brian Corcoran: not by too much; this would be a freshman-level class, so we don't assume much science background.

Jeff Elkner: *but i have students taking AP science courses, so they would be more or less on par with 1st year students at your end*

Jeff Elkner: *when could we start?*

Brian Corcoran: We're still in the process of designing the projects

Jeff Elkner: *so you are looking at next school year?*

Brian Corcoran: we were hoping this year; we don't anticipate the design taking too long.

Jeff Elkner: *that would be good, because projects work best for me at the end of the year.*

Carolann Koleci: what's your prediction for the future of python, where's it heading?

Jeff Elkner: *python seems to be entering its heyday now*

Jeff Elkner: *it is becoming more and more integrated with the Linux OS, which is also increasing in popularity*

Jeff Elkner: *it is being used for applications such as Blender, Sketch, Zope*

Jeff Elkner: *and it is finding it's way into more and more classrooms*

Brad Noyes: Okay, one last question ... Do you have any questions for us?

Jeff Elkner: *how did you come to use Python in your class design?*

Brian Corcoran: Basically, we wanted a language with a simpler syntax than what was currently being used. Also, we wanted a free, cross-platform language.

Brian Corcoran: We also wanted a language which look similar to C, MatLab, etc.

Jeff Elkner: *oops, my son just told me he has soccer practice and i need to take him.*

Brad Noyes: okay ... i think we're done.

Carolann Koleci: Thanks so much for your thoughtful replies and help!

Brad Noyes: Thanks for your time!

Jeff Elkner: *i look forward to collaborating with you in the future!*

Jeff Elkner: *cya*

Appendix C

Project Examples

C.1 Biology

Biology

Computer Simulation of Disease Propagation

References, Programming techniques

- Adapted from *Fortran 90 for Scientists and Engineers* [10], p. 609 problem 20
- Biology
- Random numbers
- Functions/subroutines
- Flow control
- Arrays

Background

One of the major uses of computer programming in biology is to simulate environments or situations which are too complex to be modeled by hand.

Problem

Suppose you need to model the spread of a epidemic (such as the recent Sars infection in Asia) on a population of N people. The spread of the disease is modeled using various parameters. Your job is to simulate the spread of the disease for a certain number of days.

Problem 1: You are given two parameters which affect the spread:

1. a "recovery" parameter: the probability that the infected individual will recover from the disease.
2. a "contact" parameter: the number of people who each infected person comes into contact with each day.

You will need to be able to run the simulation for a variable number of days, and show the results after each day.

Program

First, define the parameters you will need:

population size: an integer

number of days: an integer

recovery %: a real between 0–1

contact: an integer

Store the population as an array and use characters to denote the state of each person:

s = susceptible

i = infected

For example, if you start with one person infected, the initial array would be:

i	s	s	. . .	s	s	s

Next, define the algorithm:

If a person is infected, a random number can be compared with his or her recovery parameter to determine whether he or she will recover from the disease before contacting other persons.

Each infected person infects 'contact' number of people each day; select these people randomly from the population (including already infected people).

In order to print out the status each day, just print out each item in the array (you may want to limit the length of your array to the width of your screen).

Use subprograms for printing the status and the updating of the array.

Hints

First, write the array initialization routine as mentioned above. Next, write a subroutine to print out each element of the array.

Once this works, you need to write a function which tests if a person has recovered from the disease. This function should compare the recovery % to a random number between 0 and 1, and return *true* if the random number is less, and return *false* if the random number is greater.

The next step is to write a function to see who has been infected. Write a function to return a random number between 0 and the maximum value of the population array. The number returned will be used as an index into your population array. You don't have to worry about reinfecting people who have already been infected.

Putting it all together: Now, focus on the actual algorithm. For each day, you must print out the current population, and check if each person is infected. If they are infected, you must check to see if they recover. If they do not recover, choose another person at random to infect.

Problem 2: Problem 2 expands on problem one by adding the following parameters:

1. the "resistance" parameter: the probability that a person will not be infected by the disease upon transmission of a carrier.
2. a "recovery" parameter: the probability that the infected individual will recover from the disease before transmitting it, becoming **immune** to the disease for a duration of time.
3. a "re-susceptible" parameter: the probability that a recovered individual will become re-susceptible to infection.
4. a variable "contact" parameter: the number of people who each infected person comes into contact with each day.

Problem follow-ups:

1. Run the model several times with the same parameters. How do the simulations differ?
2. Try changing the parameters; how to they affect the model? Are there any simulations that react differently than you would expect?
3. Can you run a simulation so that there is a mix of healthy, recovered, and infected people after a year?
4. What is wrong with this model? What could be done to improve it?

5. What other types of situations could you model using similar techniques?

C.2 Physics and Mechanical Engineering

Physics/Engineering

Finite Difference

References, Programming techniques

- adapted from: http://www.funet.fi/ magi/opinnot/mpi/
- Physics/ME
- Functions/subroutines
- Flow control
- Arrays

Background

Numerical Methods is a branch of computer science which deals with problems such as these. Many problems which would be very difficult or impossible to do by hand can be solved by a computer. Numerical methods exist to solve linear systems of equations, partial differential equations, finite differences, and integration problems. As computers' power grows, scientists and engineers have been able to apply numerical methods to simulate the physical world.

Numerical Methods work by breaking up a problem into several sub-pieces which approximate the original. These simple subproblems can be accurately measured on a computer, and when pieced together, give an approximation to the solution of the original problem.

In general, numerical methods cannot find an exact answer to a problem; however, the great the number of subproblems which are solved, the greater the accuracy of the solution.

One particular area of numerical methods is finite differences. Often these problems are solved using partial differential equations; however, it is also possible to use numerical methods to solve them.

Problem 1: One-dimensional finite difference.

A heat-conducting metal wire has the left end connected to a bucket of ice water with a temperature of 0 °C and the right end connected to a kettle of boiling water of 100 °C. The ends of the wire stay at a constant temperature. The rest of the metal are room temperature 20 °C.

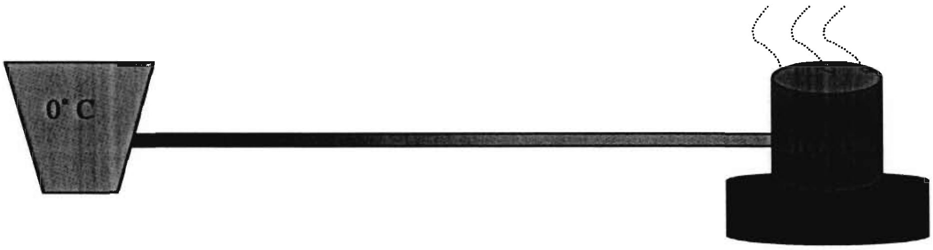


Figure C.1: Wire connecting hot and cold ends

Use the formula

$$X_{t+1}(i) = \frac{X_t(i-1) + X_t(i+1)}{2}$$

Program/Hints

- Write a function with parameters $X_t(i-1)$ and $X_t(i+1)$, to evaluate the formula. The function should return the result of the formula.
- You will need to create two arrays to hold temperatures for each slice of the metal wire for time X_t and time X_{t+1}; call the arrays `current_temp` and `next_temp`. `current_temp` will represent X_t and `next_temp` will represent X_{t+1}. Since we are dividing up the metal wire into 200 slices, make both these arrays of size 200. Initialize the values for both arrays to correspond to the initial

temperatures of the metal. So the first element should be 0, the value of the last element should be 100 and all other values should be 20.

Sample `current_temp` and `next_temp` arrays:

0	20	20	20	20	20	20	⋯	20	100

0	20	20	20	20	20	20	⋯	20	100

Each element in the array represents a slice of the metal and its current temperature. For each calculation i is represented by the current index of the array and example calculation of $i = 1$ would be

$$X_{t+1}(i) = \frac{X_t(i-1) + X_t(i+1)}{2}$$

The equation with the actual numbers would be $10 = \frac{0+20}{2}$. Insert the calculated value, 10, into `next_temp`[1].

- Iterate through the array `current_temp`, applying the formula for every element in the array, except the first element and the last element, since the temperatures at both ends of the wire will stay constant in this problem.

- In order to determine if the temperature has stabilized you need to compare the values for X_{t-1} and X_t. So after you make one set of calculations, compare the values in `current_temp` and `next_temp`. If all the values are the same, then the temperature has stabilized.

Part 2: Two-dimensional finite difference ...

Problem follow-ups

1. Determine the amount of time it takes for the temperature of the wire to stabilize, meaning there is not a temperature change from t to t+1.

2. Once the temperature has stabilized, what is the temperature of each element?

3. What are some problems with this model?

C.3 Mathematics

Mathematics

Numerical Methods for Integration

References, Programming techniques

- http://people.hofstra.edu/faculty/Stefan_Waner/RealWorld/integral/numint.html
- Mathematics
- Numerical Methods
- Functions/subroutines
- Flow control
- Random Numbers

Background

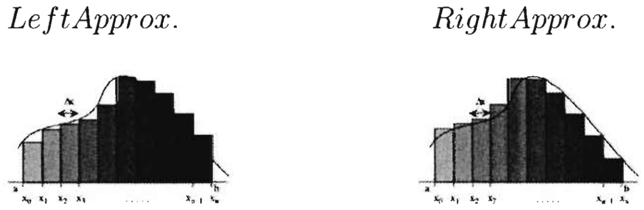
Numerical Methods: The Fundamental Theorem of Calculus gives us an exact formula for computing the integral of $f(x)$, from a to b, provided we can find an anti-derivative for f. This method of evaluating integrals is called the analytic method. However, there are times when it is difficult or impossible to find the anti-derivative of f. In these cases, it is usually good enough to find an approximate, or numerical solution.

Numerical Methods is a branch of computer science which deals with problems such as these. Many problems which would be very difficult or impossible to do by hand can be solved by a computer. Numerical methods exist to solve linear systems of equations, partial differential equations, finite differences, and integration problems. As computers' power grows, scientists and engineers have been able to apply numerical methods to simulate the physical world.

Numerical Methods work by breaking up a problem into several sub-pieces which approximate the original. These simple subproblems can be accurately measured on a computer, and when pieced together, give an approximation to the solution of the original problem.

In general, numerical methods cannot find an exact answer to a problem; however, the great the number of subproblems which are solved, the greater the accuracy of the solution.

Riemann Sums: A Riemann sum is a method of approximating the area under a curve. The area under the curve is divided up into intervals, which are either rectangular, or trapezoidal. The Riemann sum is calculated by the sum over all the intervals.



$LeftApprox.$ $RightApprox.$

Figure C.2: Example of right-hand and left-hand approximations

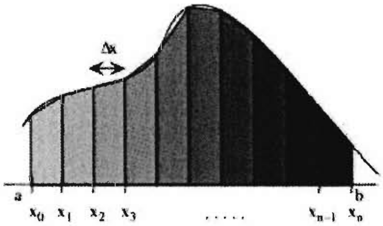


Figure C.3: Example of trapezoid

For more (perhaps better) information about Riemann sums see http://people.hofstra.edu/faculty/Stefan_Waner/RealWorld/integral/numint.html

Monte Carlo Integration: *Text below adapted from Problem 7.24, F90FSE*

Another method of approximating integrals is known as *Monte Carlo* integration. Consider a rectangle that has base $[a, b]$ and height m, where $m \geq f(x)$ for all x in $[a, b]$.

Imagine throwing q darts at the rectangle and counting the total number p that hit underneath the curve $f(x)$. For a large number of throws, we would expect

$$\frac{p}{q} \approx \frac{\text{area of shaded region}}{\text{area of rectangle}}$$

To simulate throwing the darts, generate two random numbers, X from $[a, b]$ and Y from $[0, m]$, and consider the point (X, Y) as being where the dart hits.

Problem 1: You need to calculate the integral of the function $y = x^2$ from 0 to 2. Suppose that you do not know the anti-derivative of this function, so you must find a numeric solution.

Calculate the Riemann sums for $y = x^2$, from 0 to 2. Use the left-hand sums, right-hand sums, trapezoidal sums to approximate the integral. How many iterations are needed for all three results to be within one-tenth of a unit?

Also calculate the integral for $y = x^2$ using the Monte Carlo integration.

Problem 2:

- Implement Simpson's Rule.
- Implement over 3D space.
- Read set of data point from a file and integrate over the data points.

Program/Hints:

- The trapezoidal Riemann sum is in your book.

- Write a function that evaluates $y = f(x)$ for a given point.
- Write functions that calculate the areas of a rectangle and trapezoid, given the hight, width, and, in the case of a trapezoid, the second height.
- Write a program to handle a trivial case; for example, make the number of intervals 4.

Problem follow-ups:

1. What is the numerical integral of $y = x^2$ from 0 to 2?
2. How many iterations does it take for each Riemann sum method (left hand, right hand, and trapezoidal) to converge to one-thousandth (.001) of a unit for $y = x^2$?
3. What is the numerical integral of $e^{(-x^2)}$ from 1 to 5?
4. Of the different approximation methods, which seems to be best? why?

More Information

WPI Courses that deal with Numerical Methods:

MA 3257/CS 4032 Numerical Methods for Linear and Nonlinear Systems.

MA 3457/CS 4033 Numerical Methods for Calculus and Differential Equations.

MA 4411 Numerical Analysis of Differential equations.

Glossary

abstraction The technique of generalizing a concept or problem.

algorithms A logical procedure to arrive at an end goal.

C++ An object-orient programming language based on C.

CM Chemical Engineering.

CS Computer Science. The science of solving problems with the aid of a computer.

debugging A technique to systematically find and fix problems in programs.

design recipes A problem solving technique.

emacs A text editor often used for programming.

ES Engineering Science.

FORTRAN The first programming language for numerical and scientific applications.

fruitful functions Functions which return a value.

HTML Hyper Text Markup Language. A language used to format web pages.

IDE Integrated Development Environment. An editor to ease development of programs.

Java An object-oriented programming language developed by Sun Microsystems.

libraries A collection of functions which perform a specific task (such as 3D animation).

major-based projects Projects specificly designed for a student's particular major

Maple A mathematical programming language.

MATLAB A mathematical programming application and language.

non-major A student who is not majoring, minoring, or concentrating in computer science.

NR	Not Recorded. WPI's equivalent of failing.
Python	An object-oriented programming language.
recursion	A function which calls itself; used as a form of iteration.
Scheme	A functional programming language based on Lisp.

Bibliography

[1] Chapman, S. J. *MATLAB Programming for Engineers.* Brooks/Cole, 2002.

[2] Deitel, H., Deitel, P., Liperi, J., and Wiedermann, B. *Python: How to Program.* Prentice Hall, 2002.

[3] Downey, A. B., Elkner, J., and Meyers, C. *How To Think Like a Computer Scientist: Learning with Python.* Green Tea Press, 2002.

[4] Elkner, J. Using Python in a high school computer science program: Year 1. http://www.elkner.net/jeff/pyYHS/year01/pyYHS.html, 2001.

[5] Elkner, J. Jeffrey Elkner's home page. http://www.elkner.net/, 2003.

[6] Elkner, J., Berezhny, L., and Straw, J. Using Python in a high school computer science program: Year 2. http://www.elkner.net/jeff/pyYHS/year02/pyYHS2.html, 2002.

[7] Felleisen, M., Findler, R. B., Flatt, M., and Krishnamurthi, S. *How to Design Programs: An Introduction to Computing and Programming.* MIT Press, 2001.

[8] Gauld, A. *Learn to Program Using Python.* Addison-Wesley, 2001.

[9] Goulet, J. Reconstructing a linear algebra course to serve engineering and science students. http://users.wpi.edu/~goulet/ma2071_b02/linalg_course.htm, 2002.

[10] Nyhoff, L. R., and Leestma, S. C. *Fortran 90 for Engineers and Scientists.* Prentice Hall, 1997.

[11] Python project homepage. http://www.python.org/, 2003.

[12] Radestock, M. Scheme frequently asked questions. http://www.schemers.org/Documents/FAQ/, 2003.

[13] Schools using Scheme. http://www.schemers.com/schools.html, 2002.

[14] Shannon, C. Another breadth-first approach to CSI using Python. In *SIGCSE Bulletin 35.1* (2003), pp. 248–251.

[15] van Rossum, G. Computer programming for everybody (revised proposal): A scounting expedition for the programmers of tomorrow. http://www.python.org/doc/essays/cp4e.html, 1999.

[16] VPython home page. `http://vpython.org/`, 2003.

[17] WPI proposal to revise the early undergraduate CS curriculum, 2003.

[18] WPI undergraduate catalog: 2003-2004, 2003.