Computer Science for Non-Technological Cyber Programs

Amir Rubinstein
School of Computer Science
Tel Aviv University
Tel Aviv, Israel
amirr@tau.ac.il

Abstract—The world of cyberspace revolves around the scientific and technological as well as other facets of the internet, data encryption, digital communication, signal processing and data mining. Tel Aviv University has initiated a new cyber study program for students from the non-technological disciplines. In this paper we propose an introductory Computer Science course, aimed at those students. This course exposes students, in addition to programming and several cyber-oriented topics, to general, fundamental concepts and ideas from Computer Science. The course aims at familiarizing students with the computational "culture", thus broadening their perspective on the scientific aspects of the cyber world. The Python programming language was chosen as the language used for experiencing with the course topics. This course was offered for the first time in Fall 2013. We describe the considerations in the design of the course, its content and structure. We propose several educational stand-alone modules, which can be incorporated into similar courses.

Keywords—cyber; computer science; programming; cryptography; Python

I. INTRODUCTION

The introduction and popularization of the internet, search engines, social networks, e-commerce, and numerous other services and applications into daily life of non-experts has exposed the general public to the "cyber era". The term cybernetics gained wide acceptance after a book by Norbert Wiener was published in the late 40's to denote the study of control, feedback and communication [1]. Modern use of the term cyber refers to the scientific and engineering aspects of the internet, data encryption, digital communication, signal processing and data mining. Cyber security has become a major concern of governments, security agents, financial organizations, as well as individuals around the world using the internet. However, such concerns have far-reaching consequences that lay much beyond science and engineering. Areas as diverse as ethics, law, economy, management, psychology and digital culture are all part of the cyber world these days.

In 2013, Tel Aviv University introduced, for the first time in Israel, two interdisciplinary study tracks termed "cyber". One track is focused on science and technology and will include topics in Computer Science (CS), Electrical Engineering, and Information Technology in Industrial Engineering. The other track is offered as part of the curriculum for undergraduates in Humanities, Social Sciences and Law (we refer to these as "non-technological" disciplines in this paper). This track aims at exposing students to diverse aspects of cyber, and includes courses from 6 out of 9 faculties in our university (Exact Sciences, Engineering, Social Sciences, Humanities, Law and Management). Undergraduate students from the non-technological disciplines can integrate a cluster of 7-8 courses on cyber into their study programs. For example, one class examines the legal aspects of regulating cyberspace, while another exposes students to the technical aspects of the field, including network protocols and encryption. Alongside legal and technical topics, courses cover the history and culture of cyberspace, managerial and economic facets, and issues of privacy in the digital world. Students also participate in a unique, year-long "cyber workshop," where aspects of various cyber disciplines are presented by experts in the field and discussed. A full list of the program courses appears in Table I.

Three of the courses in the non-technological track are mandatory (see Table I). One of these is a General Introduction to Computer Science (GICS) course. For most students in this program, this course will be the only one in their degree, which will expose them to programming and to some basic, fundamental computational concepts. Although there are several other courses in the program that focus on technological aspects of cyber, they concentrate on specific aspects of cyber security or computer networks; they do not teach programming; and all but one are electives. Consequently, the GICS course is expected to give students both hands-on programming experience, and a broad overview of the computational "culture" and "language".

Plenty of introductory CS courses exist worldwide. These courses differ in many aspects, such as the topics they teach, the programming language (if at all used), their target audience and background assumptions, and other structural and organizational aspects. In recent years, as the applications of computing in various scientific and non-scientific fields emerge, we witness many CS introductory courses developed for non-majors (see for example [2],[3],[4],[5]). A notable example is the emerging discipline of bioinformatics, which

This study was supported in part by a fellowship from the Edmond J. Safra Center for Bioinformatics at Tel-Aviv University.
yields many attempts to design effective educational units for students of the life sciences (for example see [6]). The innovation in this paper is that it deals with a very unique target audience of non-majors, with very specific instructional needs: students from non-technological disciplines, who wish to step out of their “disciplinary comfort zone”, and learn the insides of the cyber world, including the scientific and technological aspects of it. Therefore, none of the existing courses that we know of fits the requirements of GICS. This is mainly due to the unique combination of these three considerations:

A. The course must include basic programming

While introductory CS courses for non-CS majors sometimes avoid teaching programming, we believe that this is not the right choice here. Even though GICS students are not expected to become experts in the technological and scientific aspects of the cyber world, they are expected to gain some familiarity with the technological challenges of algorithm development, programming, and complexity issues. These challenges are much better grasped and appreciated when students struggle their way implementing an algorithm, dealing with bugs in their code, and conducting real time measurements. In addition, our experience shows that when learning includes concrete hands-on practice, computational skills are retained better, and underlying concepts are better understood. Therefore, we believe some programming, albeit basic, must be included.

B. The course must include specific topics relevant for cyber security, digital communication, and data mining

Normally, introductory CS courses do not emphasize these topics, and usually they are not included at all. GICS does put these at the spotlight, since obviously they are relevant to the program. Exposing students to these topics is expected to increase their motivation for learning and success in the program.

C. The course must expose students to some general, fundamental CS ideas and concepts

In addition to focusing on the specific cyber-oriented topics, GICS is expected to give students a glance at some CS cornerstones, which they may not encounter in other courses. Such cornerstone may include binary numbers, basic algorithms (such as binary search and various sorting approaches), recursion, notions in complexity and computability theory, and introduction to graph theory. In “pure” computer science tracks the latter topics are normally taught in more advanced courses. Obviously, we cannot use the same level of formalism as in advanced CS courses. Such topics should be introduced in a more intuitive manner that would match students’ background, and succeed to transfer the depth and beauty of such fundamental computational topics.

The combination of (A), (B) and (C) raised the challenge of developing an effective, diverse, stand-alone educational unit, exposing students from the non-technological side of the cyber world to computer science. We describe the new course, detail its topics and structure, as well as the specific design considerations raised throughout the development and teaching process. We also provide a preliminary evaluation, based on the first time this course was taught at Tel Aviv University in the fall semester 2013.

II. COURSE DEVELOPMENT AND DESIGN CONSIDERATIONS

Several years ago the school of Computer Science at Tel Aviv University initiated a revision in the introductory CS course aimed for students taking CS as their single or double major. The new course, Extended Introduction to Computer Science (EICS) is a topic based course, planned as an introduction to Computer Science rather than as an introduction to programming. EICS starts with a quick introduction to the programming language Python, and uses it to demonstrate various CS topics. A detailed description of this course appears at [7]. The development process of the GICS course was significantly inspired by EICS. One of the reasons was that EICS uses programming as a tool for implementing and explaining the ideas it plans to convey. Therefore, it spends less time and puts less focus on language specifications and syntactical issues, unlike in most other CS introductory programming courses for majors. Therefore, we found it natural to derive some of the materials and the general methodology from EICS. Furthermore, GICS inherits the “spirit” of EICS, in that both aim at providing students with a broad spectrum of ideas and concepts in computer science, and not merely to programming. However, EICS has the “privilege” of leaving some topics to advanced courses in the CS track, which students take later in their studies (e.g., computational complexity, computability and graph theory).

### TABLE I. COURSES IN THE NON-TECHNOLOGICAL CYBER PROGRAM

<table>
<thead>
<tr>
<th>Course name</th>
<th>Affiliation</th>
<th>Mandatory</th>
</tr>
</thead>
<tbody>
<tr>
<td>General introduction to Computer Science</td>
<td>Ex.</td>
<td>Yes</td>
</tr>
<tr>
<td>Fundamental technologies of cyber systems</td>
<td>Eng.</td>
<td>Yes</td>
</tr>
<tr>
<td>Signal processing</td>
<td>Eng.</td>
<td></td>
</tr>
<tr>
<td>Cyber workshop</td>
<td>Soc.</td>
<td></td>
</tr>
<tr>
<td>Law and information technology</td>
<td>Law</td>
<td></td>
</tr>
<tr>
<td>Information management in organizations</td>
<td>Mgm.</td>
<td></td>
</tr>
<tr>
<td>Data mining and neuronal networks</td>
<td>Mgm.</td>
<td></td>
</tr>
<tr>
<td>Cyber applications through learning</td>
<td>Eng.</td>
<td></td>
</tr>
<tr>
<td>models and data mining</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Principles in network theory</td>
<td>Soc.</td>
<td></td>
</tr>
<tr>
<td>Cultural history of the internet</td>
<td>Hum.</td>
<td></td>
</tr>
<tr>
<td>Data mining</td>
<td>Ex., Eng.</td>
<td></td>
</tr>
<tr>
<td>Introduction to cyber warfare</td>
<td>Soc.</td>
<td></td>
</tr>
<tr>
<td>Networks</td>
<td>Ex., Eng., Mgm.</td>
<td></td>
</tr>
<tr>
<td>Financial aspects of the cyber world</td>
<td>Soc.</td>
<td></td>
</tr>
<tr>
<td>Technological platforms in business</td>
<td>Mgm.</td>
<td></td>
</tr>
<tr>
<td>Information and power</td>
<td>Soc.</td>
<td></td>
</tr>
<tr>
<td>Internet research through the internet</td>
<td>Soc.</td>
<td></td>
</tr>
</tbody>
</table>

*We used the following abbreviations for courses faculty affiliation: Ex. – Exact Sciences, Eng. – Engineering, Soc. – Social Sciences, Hum. – Humanities, Mgm. – Management, and Law for the Faculty of Law.
GICS, on the other hand, may be the only course in which students will learn and practice general computational concepts. It therefore includes, somewhat paradoxically, topics excluded from its extended “parent”.

We now explain several considerations taken into account in the design process of the GICS course.

**Course Goals.** The course goals, as presented to the students in the course syllabus and in the first lecture are:

- To gain ability to write simple computer programs, and to understand the challenges of developing code; Put in other words, to understand “what programming is” rather than “how to program.”
- To gain basic familiarity with the scientific basis of several topics that are relevant to the cyber world.
- To be exposed to the computational “culture”, to some fundamental and elegant concepts and ideas underlying Computer Science, and to develop computational thinking skills [8].
- To gain ability to communicate effectively between people from the technological side of the cyber world.

**Students’ background.** As mentioned earlier, the GICS students come from the disciplines of Humanities, Social Sciences and Law. These students normally have no prior experience in programming or Computer Science, and our course indeed assumes no such background. However, as usually the case, some students are more “computationally oriented” than others. This is a big challenge, especially in the programming parts of this course, as some students could move forward much faster than dictated by the pace of the course.

**Level of formality.** We adjusted the level of formalism that is used to students’ background and course goals. We try to keep a lot of room for intuition and classroom discussions, and not to get drifted into a too formal or technical instruction. Nonetheless, we do insist on taking students out of their "cognitive comfort zone", in the sense that we expect them to handle abstract notions, to formulate statements in a rigorous and logical manner, and to think “algorithmically”. We believe this should foster the development of logical and algorithmic thinking skills [9].

**Programming language.** Inspired by the EICS course, as well as many other intro CS courses, we chose the Python programming language for our course as well, including the simple development environment IDLE. Python gains a lot of popularity in recent years. It is simple to learn, and it serves as an excellent tool to get into the programming experience. Furthermore, when teaching emphasized practicality rather than language syntax and technical specifications, python quickly becomes powerful enough to deliver various topics beyond programming, even in a one semester introductory course of a limited number of hours.

**Contact hours.** Our course consists of 2 weekly lecture hours spanning over 13 weeks. Unfortunately, it has no recitation or lab session, due to various constraints on the overall load of the whole program. The lack of recitations obviously makes it more difficult to develop students’ hands-on programming skills during the semester. To partially overcome this issue, almost every lecture contains some practical tasks, solved on-line in class using IDLE.

**HW assignments.** Home assignments given included both “dry”, theoretical questions, as well as “wet” programming tasks. In the latter, students were mostly asked to complete existing code, to make small modification to it for some purpose, to execute it on a different input than shown in class, or to spot and fix simple bugs. Writing code from scratch was required for only very simple problems. This is in contrast to the EICS course (and normally all introductory CS courses for majors), in which students are regularly required to implement solutions from scratch, even rather complicated ones. We found that this makes programming more accessible and less frightening to our students, and still provides good hands on experience and understanding of what programming is. In the few exercises in which students did have to write code from scratch, some of them reported feeling completely lost, and that they did not know “how to even begin” to solve the questions. However, when asked to complete or modify existing code, they seemed more motivated, less “paralyzed”, and felt that a solution is within reach. Given the above mentioned course goals, we feel this is the appropriate approach for our course.

**III. COURSE TOPICS AND STRUCTURE**

The focus of the lectures in about one third of the course (4-5 weeks) is on teaching Python (see Table II). Students are exposed to standard constructs of Python (or any other programming language for that matter), such as variables, operators, expressions, conditionals, loops, lists and functions. Later they are exposed to additional elements (such as dictionaries and recursion), as the need arises in the context of a specific topic. Week 6 introduces some basic algorithms, and running time measurements (using python’s `time` module) accompany and strengthen the distinction between logarithmic, linear and quadratic algorithms. In the second half of the course the programming skills learned serve as a vehicle to deliver, unavoidable briefly, additional topics at the heart of CS and the cyber world. However, programming skills are exercised throughout these weeks in class and HW assignments, so that at the end of the course students are expected to digest the quick dive into the programming experience of the first weeks.

By the end of the first six weeks of the course, students have learned, in addition to programming basics, three fundamental topics in CS: (1) the binary number system, (2) basic algorithms (binary search, a simple sorting algorithm and merging of sorted lists), and (3) the notion of complexity of algorithms. This leaves the second half of the course to introduce additional, more advanced topics. We find many topics relevant and fundamental enough to be included. We provide here a list of potential instructional modules that we find appropriate in this context, and briefly describe the content they may include. This list of modules can be of use to other instructors interested in developing a similar course. We later give a detailed description for the first module in the list (cryptography), based on the actual class experience.


The list of suggested modules include:

- **Introduction to cryptography.** The goal of this module is to introduce students with some basic notions in classical cryptography. It starts with the definitions of encryption and decryption. Two basic encryption approaches are presented, which are simple to understand without any background: Caesar cipher and substitution cipher. These approaches are also not too difficult to implement in code (see details later). It is important to discuss how Caesar cipher can be easily broken with a brute force approach, which is not feasible for the more general substitution cipher. It is shown how frequencies of letters in natural languages, as well as other language characteristics (e.g., common prefixes and suffixes) help partially overcome this computational barrier. The second part of this module is about secret sharing: the discrete log problem as an example for a one way function, and the Diffie-Hellman protocol. This protocol, as well as its brute force decryption are implemented in code, and students realize the latter is computationally intractable. Lastly, public key cryptography and the RSA protocol are mentioned (at a high level), as well as the underlying integer factorization problem and RSA. This protocol, as well as its brute force decryption are shown (see Fig 1). Next, students are exposed to the basics of image noise reduction: noise models such as Gaussian noise and Salt and Pepper noise, and their denoising by local means and local medians methods. Other image processing examples, which are relatively simple and intuitive are edge detection in images (using the morphological operators erosion and dilation), and image segmentation (by thresholding).

- **Complexity analysis of algorithms.** The goal is to familiarize students with the notion of complexity analysis, which is both fundamental and relevant to cyber security. In this module students are exposed to the use of the O notation in time (and possibly space) complexity analyses, in an intuitive manner. By the end of it students should realize that complexity is measured in terms of operations, not “real” time; that complexity is formulated as a function of input size; and that complexity is usually about order of growth, thus constants and low order additives are negligible. We note that this topic may not fit students who completely lack even very basic mathematical background. In such cases it is advisable to avoid using the O notation, and to explain the meaning of classes of algorithms such as logarithmic, linear, quadratic, and exponential.

```
from PIL import Image

surprise = Image.new(mode='L', size=(512,512))
matrix = surprise.load()

for i in range(512):
    for j in range(512):
        matrix[i,j] = round((i-256)**2+(j-256)**2) % 256

surprise.show()
```

Fig. 1. Example for a synthetic image, represented as a matrix of size 512X512 pixels of 256 grey hues. Each pixel represents a grey hue between pure black (0) and pure white (255). The ** operator in Python denotes exponentiation.
• **Computational complexity.** This topic includes non-formal introduction to the complexity classes \( P \), \( NP \) and \( NPC \). It starts with an explanation on how maps can be represented as graphs, followed by a definition of the graph coloring problem. Three versions of this problem serve as examples for a problem in \( P \) (2-colorability), in \( NPC \) (3-colorability), and a “trivial” problem, computationally speaking (4-colorability of planar graphs). Students are asked to formulate the necessary and sufficient conditions for 2-colorability (a graph is 2-colorable if and only if it is bipartite, i.e., contains no cycles of odd length). They are also asked to try drawing a map that is not 4-colorable, followed by a discussion on the 4-color theorem (which states that this is impossible). In addition to graph coloring, other hard problems discussed are Hamiltonian paths and traveling salesperson, as well as the discrete log and integer factorization (if not already taught as part of the cryptography module).

• **Computability and undecidability.** This module aims at exposing students to the fact that computers are not computationally omnipotent, to the theoretical limitations of computation, and to the concept of reduction. It starts with the halting problem as an example for an undecidable problem. A proof by contradiction, using the notion of an oracle is sketched. Rice theorem is mentioned, so that students get a general feeling about the type of problems that are undecidable. The “halting for every input” problem is presented, as an example for a problem that is “even harder” than the previous one. The notion of reduction is explained in this context. Then we explain how, given a hypothetical oracle for the latter problem, one could prove or disprove a mathematical conjecture, such as the Collatz \((3n+1)\) conjecture.

• **Recursion.** This is obviously one of CS cornerstones, so it is important that students are exposed to this topic. Possible simple examples include factorial, Fibonacci, merge sort and the towers of Hanoi. Underlying notions introduced are recursion depth, recursion tree, call stack and base conditions.

• **Text compression.** This module includes two compression methods: Huffman and Ziv-Lempel (ZL). In the Huffman part, students learn how to draw the Huffman tree for a given corpus, and how these enable compressing texts by assigning shorter representations for more frequent characters. In the ZL part, it is shown how compression is done regardless of a reference corpus, by exploiting repetitions in the text itself. Examples in which each of these approaches may fail to actually compress the text (yielding compression ratio >1) are shown, as well as cases in which one approach significantly outperforms the other.

• **Error detection and correction.** This module deals with one of the challenging aspects of digital communication – how to reliably transfer information over unreliable channels. Borrowing from the CS-
Privacy of biomedical data. Due to the biotechnological revolution of the recent decade, a lot of personal biomedical and genetic information is now available in digital form. Such data is often deposited in publicly accessible databases. For example, DNA sequences of 1000 individuals have been recently published [12]. This raises some unexpected privacy issues. For instance, it was shown that by analyzing only 30 markers from the Y chromosome of an unknown individual from the USA, one can determine this individual’s last name with over 20% success [13]. This module discusses some related issues.

We note that the topics of computer networks and cyber security, which are highly relevant to this program, are not included in this list, since they are the raison d’être of the other mandatory technological course in the program (Fundamental technologies of cyber systems). In the first cycle of GICS, we were not able to include all these topics due to time constraints (see Table II for topics included). Supporting instructional material (Python code and slides) for most of these topics (even those that were not actually taught) were prepared and are available upon request.

In the first cycle of the GICS course, cryptography was among the modules taught. Below is a detailed description of how this topic was transferred in the course. The instruction of this topic spanned over 2 lectures (4 hours in total).

We started this module with a short introduction, explaining the three main challenges of digital communication: security (cryptography), cost (compression) and reliability (error detection codes). After basic notions in cryptography were taught, we introduced the Caesar (offset) cipher. We showed a simple function in Python that, given a text and an offset, returns the encrypted text, using the modulo (%) operator. We then showed how one can easily break this simple code, by trying all possible offsets. Students realized that human intervention is needed to identify the correct offset. This can be partly overcome if we provide the computer a dictionary of common words and a threshold for matches required to declare a success in decryption. We showed an implementation of this approach, and executed several trials on real texts. There was an interesting and fruitful discussion in class about the words that should be included in that dictionary, the effect of a too small or too large a threshold, and how prior knowledge one has on the original text may come handy.

We then moved on to substitution ciphers, and explained that they are a generalization of the previous method. We explained why a brute force decryption would not work in this case, as the number of possible ciphers is n! when the alphabet size is n. We estimated running time for English lowercase letters alphabet (26! Possible substitution ciphers), by a computer capable inspecting 10^10 ciphers a second. The result, about 10^19 years, seemed to impress students and gave them a good feeling that this limitation is not only theoretic. Next, we discussed how knowledge on the letter frequencies may assist us to overcome this computational barrier. The programming part traced the whole process of encryption and decryption, with the following functionalities shown and explained:

1. Encryption: creating a random substitution cipher of the English lowercase letters and the space character. The cipher was created using Python’s random module, and it was represented as a dictionary (Python’s dict class). Then it was used for encrypting some text.

2. Corpus analysis: downloading a real text from the internet (we used “The Adventures of Tom Sawyer,” “Romeo and Juliet,” and the front page of the New York Times). Then cleaning the text from non-lowercase English letters or spaces, and counting letter frequencies.

3. Decryption: using part 2 and the acquired letter frequencies to decrypt a given ciphertext, encrypted earlier by the instructor.

Students realized that small differences between texts in the order of letter frequencies prevent the third part from being a fully automatic process. Specifically, the three corpuses we used agreed only on the two most frequent letters (‘e’ and ‘t’). Indeed, the first decryption attempt, in which we substituted the 5 most frequent letters in one of the corpuses, did not yield a reasonable English text. We therefore decided to use only the 2 most frequent letters in all corpuses as a starting point for decryption. Then students suggested, at different stages of this process, possible ways to fill out missing letters or words. Fig. 3 illustrates part of this activity as conducted in class, in a very interactive manner. This led to several reflections on how to improve our approach. For example, by identifying common English letter sequences (such as ‘ing’, ‘wh’, ‘ee’, etc.), common words (such as ‘the’), or using knowledge on letter placement within words (for example that proper English words do not start with ‘x’).

The rest of the time (the 4th hour out of 4 dedicated to cryptography) was devoted to secret sharing over an unsecure communication channel using the Diffie-Hellman protocol, and the underlying discrete log problem. We do not elaborate on this further here. Due to time constraints, integer factorization and RSA protocol were only briefly mentioned, without getting into much detail. However it was emphasized that in both cases, hardness of computational problems is exploited.

In the HW assignment, students were asked to decrypt two texts: a short message encrypted by a Caesar cipher (the offset was unknown to them), and a longer text (Nelson Mandela’s speech at treason trial, 1964) encrypted by a random substitution cipher. In the first problem, students realized a dictionary of common words is of no use, since the text is rather short. Students reported the second problem enjoyable, since they had to act like “detectives” and look for the best hints at every stage of the solution.
IV. PRELIMINARY EVALUATION AND SUMMARY

The course was taught for the first time in the fall semester 2013 at Tel Aviv University, as part of the newly opened cyber program. 12 students participated, from the faculties of Social Sciences and Humanities. The average in the final exam was 71. The end-of-course evaluations were submitted by 10 students before the final exam took place. Due to the small number of students of the first offering, these results have limited statistical significance. However, they give some indication that students had positive feelings about the course. On a Likert scale of 1 (low) to 7 (high), students were required to report on their agreement with several statements, including: (a) the lectures were interesting, and (b) the lectures were intellectually challenging. The means from the Fall 2013 offering of GICS, as well as the means of all courses in the non-technological cyber program courses were:

<table>
<thead>
<tr>
<th></th>
<th>GICS</th>
<th>Cyber</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall satisfaction</td>
<td>5.78</td>
<td>5.52 (s=1.19)</td>
</tr>
<tr>
<td>Interest</td>
<td>5.89</td>
<td>5.48 (s=1.34)</td>
</tr>
<tr>
<td>Intellectual challenge</td>
<td>6.56</td>
<td>5.43 (s=1.24)</td>
</tr>
</tbody>
</table>

Based on our experience in teaching this course and on students’ feedback, we strongly feel that the course contributed to students’ understanding of basic notions in Computer Science, especially those relevant for the cyber world. We plan additional offerings of the course in the coming years, which will allow a methodological assessment of the course goals and learning outcomes.  

An introductory CS course for non-technological cyber students (or non-majors, for that matter) is an opportunity to design students’ attitudes towards Computer Science and the technological facets of the digital world. Therefore, it is important for such courses to include affective learning outcomes, not only cognitive ones. For example, programming experience should be positive and satisfying, at the expense of syntactic details and teaching additional programming construct. This is likely to increase students’ tendency to be engaged with programming after the course ends. Deeper computational topics and applications should be delivered in a “friendly” manner, avoiding a too technical and formal instruction. This way, students are more likely to appreciate the beauty of these topics.

We expect similar interdisciplinary study programs on the various facets of the cyber world, as was initiated in our university, to spread worldwide in the near future. We hope that the pilot described in this paper could be useful to instructors in such institutes, planning to teach similar courses.

ACKNOWLEDGMENT

We thank Benny Chor for a critical review of this manuscript, for contributing material for the course, and for helpful discussions.

REFERENCES


