UNIVERSITY OF NEBRASKA AT OMAHA
COURSE SYLLABUS

Department and Course Number	CSCI 1200
Course Title	Computer Science Principles
Course Coordinator	Harvey Siy
Total Credits	3
Repeat for Credit?	No
Date of Last Revision	May 7, 2013

1.0 Course Description Information

1.1 Catalog description:

This course introduces students to the foundational principles of computer science. It aims to help students learn the essential thought processes used by computer scientists to solve problems, expressing those solutions as computer programs. The exercises and projects make use of mobile devices and other emerging platforms.

1.2 Prerequisites of the course:

MATH 1310 (or equivalent)

1.3 Overview of content and purpose of the course:

This is an implementation of the CS Principles course (http://www.csprinciples.org) proposed by the College Board and the National Science Foundation as a new AP course in Computer Science. The purpose is to broaden participation in computing and computer science by providing a fresh approach to teaching computing, focusing on the notion of computational thinking, and imparting the following themes (a.k.a, "Big Ideas"):

1. **Creativity**: Computing is a creative activity.
2. **Abstraction**: Abstraction reduces information and detail to facilitate focus on relevant concepts.
3. **Data**: Data and information facilitate the creation of knowledge.
4. **Algorithms**: Algorithms are used to develop and express solutions to computational problems.
5. **Programming**: Programming enables problem solving, human expression, and creation of knowledge.
6. **Internet**: The Internet pervades modern computing.
7. **Impact**: Computing has global impacts.

Furthermore, due to the increasing popularity of mobile applications, the use of mobile devices makes the course relatable and attractive to students who may not otherwise be interested in computing. Note that the choice of programming platform may change over time as newer technologies become available.

1.4 Unusual circumstances of the course.

None.

2.0 Course Justification Information

2.1 Anticipated audience / demand:

This course is applicable for students in all majors who have no prior background in computer programming and who want to learn how to solve problems using computational techniques.

2.2 Indicate how often this course will be offered and the anticipated enrollment:

This course will be offered regularly in the Fall and Spring semesters with an anticipated enrollment of around 60 students per semester.

2.3 If it is a significant change to an existing course, please explain why it is needed:
This is a new course. It is intended to broaden participation in computing. It is also intended to supplement existing preparatory courses being used to get students ready for the more rigorous computer programming courses by focusing more on basic computational problem solving techniques rather than mastery of a certain language or technology.

3.0 List of performance objectives stated in learning outcomes in a student's perspective:

3.1 Students will understand the societal need for continued computing innovations.
3.2 Students will be able to express algorithms in a well-defined and unambiguous manner.
3.3 Students will map practical problems to a computational solution.
3.4 Students will develop appropriate abstractions to manage problem complexity.
3.5 Students will use programming as a creative tool.
3.6 Students will learn teamwork implementing a small group project.

More detailed learning objectives and evidence statements are available from the CS Principles website.

General Education Student Learning Objectives

Describe how the course meets the Student Learning Outcomes

Natural and Physical Sciences: SLO #1: *Demonstrate a broad understanding of the fundamental laws and principles of science and interrelationships among science and technology disciplines.*

Increasingly, computer science is getting recognition as the third pillar of science alongside theory and experimentation (PITAC, 2005). It enables the study of theoretical models of phenomena too complex, costly, hazardous, vast or small for experimentation. All disciplines of science stand to benefit from high-resolution model predictions, theoretical validations and experimental data analysis brought about by advances in computing and computing technology. Hence computer science is indispensable to 21st century scientific endeavors. CSCI 1200 serves as an introduction and broad overview to the field of computer science.

In the process, the course examines how computation pervades and enables modern scientific discovery by supporting data collection, modeling, analysis and visualization, a new paradigm known as computational and data-enabled science and engineering. Much as experimentation helps students connect theory to natural phenomena, the design and solution of computational models serves a similar purpose for furthering the understanding of scientific theory. The projects and exercises in the course are geared towards helping students develop the basic skills needed for building computational models.

Natural and Physical Sciences: SLO #2: *Demonstrate a broad understanding of various natural phenomena that surround and influence our lives.*

While many consider computer science to be an artificial science because it studies phenomena surrounding computers and related digital technologies, in its essence, computer science studies information processes both artificial and natural (Denning, 2005). Information processes take data inputs and produce outputs just as physical and

biological processes take stimuli and produce responses. There has been in increasing realization that many natural phenomena are, fundamentally, information processes. DNA encodes information about living organisms, and quantum electrodynamics had been described as nature's computational method for combining quantum particle interactions (Denning, 2007). To study information processes, there is a need to study the foundational computing principles that underlie these processes. These principles are the central focus of CSCI 1200.

Furthermore, scientists have come to increasingly rely on computational models to study and describe complex phenomena in climate science, particle physics, system biology, etc. Many problems in these fields are not amenable to theoretical solutions and can only be addressed through large scale computations and simulations. Due to its crucial role, computational errors arising from faulty programming can produce incorrect results and have been known to lead to retractions (Merali, 2010). Formal training of future scientists in the computer science disciplines can reduce the risk of such errors. CSCI 1200 provides a starting point for studying the principles of sound software development practices.

Natural and Physical Sciences: SLO #3: *Describe how scientists approach and solve problems including an understanding of the basic components and limitations of the scientific method.*

A foundational component of the course is the introduction of computational thinking (Wing, 2006). Computational thinking is broadly defined as the process by which computer scientists approach and solve problems. Computational thinking is a way of solving problems by mapping a natural problem into an algorithmic model that is amenable for computational solution. This is a form of empirical inquiry (Newell and Simon, 1976) similar to how scientists would approach a problem by devising hypotheses and designing experiments to validate the hypothesis. The implementation of the program on a computational device becomes the experiment that validates the algorithmic model.

Natural and Physical Sciences: SLO #4: *Solve problems and draw conclusions based on scientific information and models, using critical thinking and qualitative and quantitative analysis of data and concepts in particular to distinguish reality from speculation.*

CSCI 1200 gets away from traditional pedagogical models that introduce computer science by teaching students how to use a particular computing technology or how to write programs in a particular language. Instead, the course seeks to instill the discipline of computational thinking in students. Computational thinking complements other critical thinking skills by helping students think about how to make use of increasingly ubiquitous computing devices (PCs, mobile devices, etc.) to aid them in solving everyday problems. An overarching goal of the course is to help students gain an increased awareness of how computing impacts their lives and the world around them. It emphasizes that computer science is not an isolated field, but rather has relevance in everyday life, and when combined with other math and science disciplines, becomes a powerful vehicle for addressing relevant problems. In particular, the semester project of the course will provide first-hand problem-based learning experiences through developing mobile applications that address real-life problems.

References

(Denning, 2005) Denning, Peter J. (2005). "Is computer science science?" *Communications of the ACM*, 48(4): 27-31.

(Denning, 2007) Denning, Peter J. (2007). "Computing is a natural science." *Communications of the ACM*, 50(7): 13-18.
(Merali, 2010) Merali, Z. (2010). "Computational science: ...Error," *Nature*, vol. 467, no. 7317, pp. 775–777, October 2010. [Online]. Available: http://www.nature.com/news/2010/101013/full/467775a.html
(Newell, Perlis and Simon, 1967) Newell, Allen, Alan J. Perlis, and Herbert Simon. (1967). "What is computer science?" *Science*, 157(3795): 1373-1374.
(Newell and Simon, 1976) Newell, Allen, and Herbert A. Simon. (1976). "Computer science as empirical inquiry: Symbols and search." *Communications of the ACM*, 19(3): 113-126.
(PITAC, 2005) President's Information Technology Advisory Committee. (2005). "Computational science: ensuring America's Competitiveness." http://www.nitrd.gov/pitac/reports/20050609_computational/computational.pdf
(Wing, 2006) Wing, Jeannette M. (2006). "Computational thinking." *Communications of the ACM*, 49(3): 33-35.

4.0 Content and Organization Information

4.1 List the major topics central to this course:

- 4.1.1 Societal impact of computing (1.5 hours)
- 4.1.2 Computational thinking (1.5 hours)
- 4.1.3 Algorithms (3 hours)
- 4.1.4 How computers work:
  - 4.1.4.1 The von Neumann architecture (1.5 hours)
  - 4.1.4.2 Data and bits (1.5 hours)
- 4.1.5 Exploring programming
  - 4.1.5.1 Variables and data manipulation (3 hours)
  - 4.1.5.2 Decisions, loops and logical reasoning (3 hours)
  - 4.1.5.3 Simple data structures (6 hours)
- 4.1.6 Designing larger programs
  - 4.1.6.1 Modularization (3 hours)
  - 4.1.6.2 Stepwise refinement (3 hours)
- 4.1.7 Writing applications
  - 4.1.7.1 Web applications (6 hours)
  - 4.1.7.2 Mobile applications (6 hours)
  - 4.1.7.3 Big data (3 hours)
  - 4.1.7.4 Graphics and multimedia (3 hours)

Table 1 shows how the course topics map to the Student Learning Objectives (SLOs) for science:

1. Demonstrate broad understanding of fundamental laws and principles of science and interrelationships among science and technology disciplines
2. Demonstrate a broad understanding of various natural phenomena that surround and influence our lives.
3. Describe how scientists approach and solve problems including and understanding of the basic components and limitations of the scientific method.
4. Solve problems and draw conclusions based on scientific information and models, using critical thinking and qualitative and quantitative analysis of data and concepts in particular to distinguish reality from speculation.

Table 1: Mapping between course topics and student learning objectives for science.

Content	Relevant Student Learning Objectives
4.1.1 Societal impact of computing	SLO #1: Emphasizes that computing is not just an isolated

(1.5 hours)	activity for hackers but has strong connections with other STEM disciplines and that the integration has great potential for addressing diverse community-relevant problems. SLO #2: Explores how technology connects and influences the activities of 21st century society.
4.1.2 Computational thinking (1.5 hours)	SLO #3: Introduces the foundational component of the course, computational thinking. Computational thinking is the process by which computer scientists approach and solve problems by mapping a natural problem into an algorithmic model that is amenable for computational solution.
4.1.3 Algorithms (3 hours)	SLO #1: Algorithms are the step-by-step solution to a computational problem. This topic introduces algorithms as a way of describing information processes so that they can be studied further. SLO #3: Introduces algorithms as the creative expression of computational thinking.
4.1.4 How computers work (3 hours)	SLO #2: Studies the phenomena surrounding computers, how they work internally, particularly the common von Neumann architecture. Understanding this phenomenon provides a frame of reference for understanding how algorithms run and why they are expressed that way.
4.1.5 Exploring programming (12 hours)	SLO #2: Programs are algorithms realized in a formal language that computers can understand. The execution of programs on a computer and the inevitable debugging activity further helps students understand the phenomenon of computing.
4.1.6 Designing larger programs (6 hours)	SLO #3: Introduces the important computational thinking concept of abstraction, which enables students to describe larger problems and solve them with larger programs.
4.1.7 Writing applications (18 hours)	SLO #1: Introduces various modern computing technologies (web, mobile, graphics, databases) and provides a framework for integrating them together to solve STEM-related problems. SLO #1: Studies the complex information processes resulting from the interplay of devices, networks and data. SLO #2: Examines how technology enables scientific discovery by supporting data collection and analysis, also known as cyber-enabled discovery and innovation. SLO #2: Develops skill in creating meaningful models to understand natural phenomena and solve real-world problems. SLO #3: Demonstrates how problems can be systematically and methodically solved by the combining the strengths of diverse technologies.

	SLO #4: Introduces the technologies that are the basic ingredients for “building a smarter planet”. The technologies are likely to change over time as new ones emerge that better suit the learning objectives of an increasingly sophisticated student population. (The contributions and objectives of the current technologies are explained below.)
4.1.7.1 Web applications (6 hrs)	SLO #4: Introduces the cloud which enables flexible computing resources and provides centralized access to data and applications. SLO #3: Studies how the web enables collaborative problem solving.
4.1.7.2 Mobile applications (6 hrs)	SLO #4: Introduces application development on mobile devices, which are practically portable computers with state-of-the-art sensors for collecting data about its physical environment. SLO #2: Sensor data of environment enables study of the natural phenomena that surrounds us.
4.1.7.3 Big data (3 hrs)	SLO #4: Solves real-world problems through the development of solutions that are grounded in data rather than speculation. SLO #2: Analysis of scientific data facilitates cyber-enabled discovery.
4.1.7.4 Graphics and multimedia (6 hrs)	SLO #4: The foundation for visual data analytics, which is crucial to analyzing and understanding data.

5.0 Teaching Methodology Information

5.1 Methods:

This course will be held in a hybrid lecture/lab format every week. The lecture describes essential computational concepts while the lab enforces the concepts through active learning techniques.

The course uses two programming languages, starting off with a visual programming language and transitioning to a textual language in the latter half of the semester. The choice of language may change over time with adoption of newer programming platforms.

For the initial offerings of this course, the first part uses the App Inventor language (http://appinventor.mit.edu), a simple graphical language used to create applications running on the Android platform. App Inventor programs are expressed as a collection of blocks, each corresponding to some statement. Like similar languages such as Scratch and Alice, App Inventor was designed to have a low barrier of entry for novices to write programs without becoming frustrated with program syntax.

The second part of the course uses Python, an interpreted procedural language. This was intended to give students a feel for reading and writing programs in textual languages, facilitating a smoother transition into conventional languages like Java.

5.2 Student role:

Students are expected to attend the lectures and participate in class discussions. They are required to do all homeworks and assigned readings. To spur creativity, students will also propose and implement a small team-based project.

6.0 Evaluation Information

6.1 Describe the typical types of student projects that will be the basis for evaluating student performance:

Homework assignments will typically be small problems requiring programming solutions as well as short writing assignments or reports discussing various societal impacts of computing.

The term project will be an application involving mobile devices. Students will give two oral presentations on the project, once at the onset, describing their proposal, and then at the end, presenting the finished project.

6.2 Describe the typical basis for determining the final grade (e.g. weighting of various student projects):

Homework assignments/laboratories: 50%
Term project: 20%
Midterm exam: 10%
Final exam: 20%

6.3 Grading type:

Percent	**Grade**	**Percent**	**Grade**
97 – 100	A+	77 – 79	C+
93 – 96	A	73 – 76	C
90 – 92	A–	70 – 72	C–
87 – 89	B+	67 – 69	D+
83 – 86	B	63 – 66	D
80 – 82	B–	60 – 62	D–
		0—59	F

7.0 Resource Material Information

7.1 Textbooks and/or other required readings used in course:

Hal Abelson, Ken Ledeen, Harry Lewis. (2010). Blown to Bits: Your Life, Liberty, and Happiness After the Digital Explosion. Addison-Wesley. http://www.bitsbook.com.

Allen Downey. (2012). Think Python: How to Think Like a Computer Scientist. Green Tea Press. http://www.greenteapress.com/thinkpython.

David Wolber, Hal Abelson, Ellen Spertus, and Liz Looney. (2011). App Inventor: Create Your Own Android Apps. O'Reilly Media, Inc. http://www.appinventor.org/projects.

7.2 Other student suggested reading materials:

John Conery. (2010). Explorations in Computing: An Introduction to Computer Science. CRC Press.

7.3 Current bibliography and other resources:

J. Glenn Brookshear. (2011). Computer Science: An Overview, 11th Edition. Addison-Wesley.

Matthias Felleisen, Robert Bruce Findler, Matthew Flatt, Shriram Krishnamurthi. (2012). How to Design Programs, 2nd Edition. MIT Press. http://htdp.org.

Jerry Lee Ford. (2008). Scratch Programming for Teens. Course Technology PTR.

Tony Gaddis. (2011). Starting Out with Alice, 2nd Edition. Addison-Wesley.

Tony Gaddis. (2012). Starting Out with Programming Logic and Design, 3rd Edition. Addison-Wesley.

Mark J. Guzdial, Barbara Ericson. (2012). Introduction to Computing and Programming in Python, 3rd Edition. Prentice Hall.

Michael Kölling. (2009). Introduction to Programming with Greenfoot: Object-Oriented Programming in Java with Games and Simulations. Prentice Hall.

Jeff Kramer. (2007). Is abstraction the key to computing? Communications of the ACM 50, 4, April 2007, pp. 36-42.

Casey Reas, Ben Fry. (2010). Getting Started with Processing: A Quick, Hands-on Introduction. O'Reilly Media.

David Reed. (2010). A Balanced Introduction to Computer Science, 3rd Edition. Prentice Hall.

8.0 Other Information:

8.1 Accommodations statement:

Accommodations are provided for students who are registered with UNO Disability Services and make their requests sufficiently in advance. For more information, contact Disability Services (EAB 117, Phone: 402.554.2872, TTY: 402.554.3799) or visit the web at http://www.unomaha.edu/disability.

8.2 Other:

8.3 Author(s):

Harvey Siy and Jong-hoon Youn

9.0 Computer Science Accreditation Board (CSAB) Category Content (class time in hours):

CSAB Category	*Core*	*Advanced*
Data structures	10	0
Computer organization and architecture	5	0
Algorithms and software design	45	0
Concepts of programming languages	5	0

10.0 Oral and Written Communications:

Every student is required to submit at least 1 written report (not including exams, tests, quizzes, or commented programs) to typically 10 pages and to make 2 oral presentations of typically 10 minutes duration.

11.0 Social and Ethical Issues:

Throughout the course, we will introduce the social impacts of computing by including discussion from the text "Blown to Bits" [Abelson, et al. 2010]. We will also connect lab exercises to real-world problems.

12.0 Theoretical content:

Please list the types of theoretical material covered, and estimate the time devoted to such coverage.

Algorithm analysis – 3 hours
Reasoning about programs – 3 hours
Abstraction and modeling – 3 hours

13.0 Problem analysis:

Please describe the analysis experiences common to all course sections.

We will focus on helping students develop a mindset for problem solving using computational thinking.

14.0 Solution design:

Please describe the design experiences common to all course sections.

Students will have the opportunity to design their own mobile applications and implement them using App Inventor and Python.

CHANGE HISTORY

Date	***Change***	***By whom***	***Comments***
8/8/2012	Initial special topics course draft	Siy	
8/9/2012	Added sections 8-14.	Siy	
8/27/2012	Revised into a regular course proposal.	Siy	
9/21/2012	Revised content; more applications	Siy	
10/10/2012	Revised catalog description, added prerequisite, reorganized bibliography	Siy	
10/11/2012	Clarified in Section 6 how written and oral communication requirements (Section 10) fit in the evaluation process.	Siy	
5/7/2013	Added Science Student Learning Objectives so that course can be used as a general education course.	Siy	

This table is used to relate performance objectives (in section 3 of the syllabus) to the ABET program outcomes. List the appropriate performance objectives in the left column. In the body of the table, use S and X to indicate the relationship between the performance objective and the ABET program outcomes. Leave cells blank if there is no relationship. Add additional rows to the table as needed.

S – Strong relationship
X – Contributing relationship

Course objective	**(a) knowledge of discipline**	**(b) analyze problem, define requirements**	**(c) design and implement solution**	**(d) function on a team**	**(e) ethical issues**	**(f) communicate effectively**	**(g) analyze impact of computing**	**(h) continued professioanl development**	**(i) Current techniques and tools**	**(j) apply foundations**	**(k) apply design and development principles**
3.1	**X**				**S**		**S**	**S**			
3.2	**S**		**S**								**S**
3.3		**S**								**X**	
3.4		**S**	**S**								**X**
3.5			**S**						**S**		
3.6				**S**		**S**					