Course Submission Form

Instructions: All courses submitted for the Common Core must be liberal arts courses. Courses submitted to the Course Review Committee may be submitted for only one area of the Common Core and must be 3credits/3contact hours. Colleges may submit courses to the Course Review Committee before or after they receive college approval. STEM waiver courses do not need to be approved by the Course Review Committee. This form should not be used for STEM waiver courses.

Form ID CCOREFORM1085170953002 **Version No.** 42.002 **Created by** Vickery, Christopher

Created on 2019-09-25T08:29:46 **Last Updated on** 2019-09-25T08:59:37 **Status Updated on** 2019-10-18T10:02:10

Current Status Approved **Course Selected:** **Subject** CSCI (CSCI - Computer Science) **Catalog Nbr** 111

Course Revision & College

Form Submission Revised Submission **College** Queens College

Please describe revisions that have been made to this course The sample syllabus from the previous submission has been replaced with a revised and expanded version that addresses the concerns expressed by the reviewers. (File name is CSCI-111_2019-09-24.pdf.) Among other changes, this syllabus includes a link to an extensive repository of questions from previous exams so that reviewers (and students!) can get a good sense of the scope and level of learning required of students.

To address reviewers' concerns about assignment details, we are providing two practices sets that are part of this term's actual (not sample) syllabus given to students. Set-1 is on nested loops and is given early in the term; Set-3 is part of the "extensive" set mentioned above, given at the end of the term.

The SLO justifications have not been changed because the reviewer feedback was positive about the original versions.

Course Data

Course ID 005654 **Subject** CSCI (CSCI - Computer Science) **Catalog Nbr** 111

Catalog Status Approved **Contact Hours** 4 **No. of Credits** 3

CourseTitle Introduction to Algorithmic Problem-Solving

Course Description Introduction to the principles of algorithmic analysis and computational implementation. Topics include implementation methodologies, including choice and use of data types, objects, classes, and methods; control structures; basic data structures including arrays; procedures and functions; parameters and arguments; scope and lifetime of variables; input and output; Written documentation describing algorithms and identification and correction of algorithmic implementations.

Department Computer Science

Pre-Requisites/Co-Requisites PREQ/COREQ: MATH 120 OR 141 OR 151 with min grade of C-

Course Syllabus [Attachment Filename(s)]
CSCI-111_2019-09-24.pdf
CSCI-111_Set-1.pdf
CSCI-111_set-3.pdf

Location(Required or Flexible) and Learning Outcomes

REQUIRED		FLEXIBLE
English Composition		World Cultures & Global Issues
Math & Quantitative Reasoning		US Experience in its Diversity
Life and Physical Sciences		Creative Expression
		Individual and Society
	☑	Scientific World

Learning Outcomes: Questions	Learning Outcomes: Responses

*** 1. Gather, interpret, and assess information from a variety of sources and points of view.**	While implementing any algorithm in computer code, the programmer receives, interprets, and integrates information (and attendant uncertainties) from at least three different perspectives, represented by the three principal partners in the creation of software: the owner of the project who specifies a problem to be solved, the client who will eventually use the software, and the programmer who creates the solution. Whenever an algorithm is presented in class, the owner's perspective is shown first, to define the problem. The next step is the analysis of the client's perspective, to envision the desired user experience. Finally, the main discussion follows the programmer's perspective, where key design concerns are correctness, practicality and efficiency of an implementation. The solution is synthesized by considering and resolving the competing claims of the three perspectives. Every lab exercise in the course is developed under this set of perspectives. In general, the lab instructor lays out a problem from the owner's perspective. Students then form groups to discuss interaction between the solution and users. Individual students come to assume various roles in this discussion. These two stages lead to understanding the implementation as a synthesis of the needs of the three partners. Finally, the hard work of coding an actual implementation is carried out by students on an individual basis, but the requirements of the owner and clients must be honored.

*** 2. Evaluate evidence and arguments critically or analytically.**	The essential core of the course is the creation and implementation of algorithms. Every algorithm is a sequence of exercises in analytical thinking, the core of which has the strength and analytical rigor of mathematical reasoning. Every program ultimately represents a mathematical function. This function, while expressed in a distinctive form, remains amenable to mathematical reasoning. While this analytical reasoning constitutes the daily work of a professional programmer, in a scope much broader and at a level much higher than can be attained in the course, the course offers a qualitatively authentic experience of this reasoning, through a collection of coding exercises. These exercises require students not only to analyze the structure of digital data and algorithms for its manipulation, but also to analyze anomalies in the process of generating correct code. Evidential reasoning is precisely the cognitive skill developed by this part of the course. Key arguments are guided by the following sorts of questions: What data defines the input to a problem? What data should be computed to define the solution? For each computational step coded, what exactly is its effect? Under which assumptions is the solution plausible? What is the set of input data on which the completed program ought to be tested? For each test point, what is the expected behavior of the program? For each observed behavior of the program in testing, what conclusion is to follow any the discrepancy between the observed and desired behavior? How will the correctness, efficiency, and user experience of the solution change if original assumptions are altered in some specific way?

*** 3. Produce well-reasoned written or oral arguments using evidence to support conclusions.**	In all lab exercises, while discussing their code, especially while focusing on planning the solution, analyzing its correctness, evaluating user experience, and analyzing results of testing, students take various positions in oral arguments, convincing each other and the lab instructor of the veracity of the claims they make about the code and the behavior demonstrated by the code. In their exams and preparation for exams, students must develop arguments in writing, in response to the evidence presented in exam questions. Such questions could give a piece of C++ code, which students have to understand without the opportunity to run it, i.e., they have to predict behavior of the code by reasoning from the code itself (and perhaps from selected evidence of its behavior included in the question.) Conclusions inevitably may fall in a broad spectrum, from exactly correct on one end, to hopelessly flawed on the other. The only way to arrive at the answer is to construct a well reasoned argument.
4. Identify and apply the fundamental concepts and methods of a discipline or interdisciplinary field exploring the scientific world, including, but not limited to: computer science, history of science, life and physical sciences, linguistics, logic, mathematics, psychology, statistics, and technology-related studies.	The fundamental concept in the discipline of Computer Science is that of an algorithm. An algorithm is a formal procedure designed to solve a problem. Algorithms are the central topic of CSCI 111. In every lecture of the course, every lab session, every homework problem and exam problem, students learn how to formulate problems in ways amenable to algorithmic analysis, how to devise appropriate algorithms and how to write these algorithms as concrete C++ programs.
5. Demonstrate how tools of science, mathematics, technology, or formal analysis can be used to analyze problems and develop solutions.	The fundamental primitives of algorithmic thinking, which appear as the basic tools in virtually all computer languages and computing systems, are: variables, decisions, loops, functions, recursion, arrays, data structures, and external storage. In the CS 111 course, these primitives are studied individually, in sequence as shown on the sample syllabus. Some of the homework and exam problems are designed to focus on a single tool, introducing gradually constructions where the tools are at their most useful, by being applied in combination. By the end of the course, and in final exam questions, students are expected to be able to write algorithms that combine all of these fundamental tools.

6. Articulate and evaluate the empirical evidence supporting a scientific or formal theory.	
7. Articulate and evaluate the impact of technologies and scientific discoveries on the contemporary world, such as issues of personal privacy, security, or ethical responsibilities.	
8. Understand the scientific principles underlying matters of policy or public concern in which science plays role.	The CS 111 course is the boundary line where students cross from the position of a passive user of algorithms to the position of an active creator. When students begin the course, they know that all modern technology is driven by software (i.e. algorithms in action). Like almost all people, they are happy to use this technology even though they are aware that software can exhibit errors, vulnerabilities, biases and inconvenient behavior. In this course, students begin to learn just how the ropes and levers are operated to move their world, facing their responsibility as creators of algorithms. In lectures, these issues are unavoidably discussed. Beyond discussion, all lab assignments in CSCI 111 require students to understand, accept and practice a set of standards designed to give some protection against errors, vulnerabilities, and biases. In the course, all programs are subjected to debugging and testing, to eliminate errors. Effective documentation must be written to accompany their code, to enable interested non-experts to understand the algorithm behind it. Knowledge of the algorithm and clear, but rigorous specification of its functionality serve to predict impact and detect errors or hidden biases. Solutions submitted to programming assignments must offer appropriate and convenient user interfaces. By putting these standards in effect and observing their impact, students begin to prepare for a lifetime of attention to the complex relationship between the abstract scientific concept of an algorithm on the one hand and its concrete embodiment of deployed software on the other hand.
A. If there is a change to the course title, what is the new course title?	PLEASE NOTE: this courses is currently designated as an MQR course under the STEM Variant rule. If approved, the SW designation will replace the MQR designation. This change will impact no students because the course has other MQR courses as a prerequisite.
B. If there is a change to the course description, what is the new course description?	

C. If there is a change to the pre-requisites and/or co-requisites, what are the new pre-requisites and/or co-requisites?	

Chair (Approver) Comments

Comments The course is approved.