C S 49: FOUNDATIONS OF COMPUTER PROGRAMMING

Foothill College Course Outline of Record

Heading	Value
Effective Term:	Summer 2021
Units:	2
Hours:	2 lecture, 1 laboratory per week (36 total per quarter)
Advisory:	MATH 105 or equivalent; concurrent enrollment in ESLL 125 or ENGL 209.
Degree & Credit Status:	Degree-Applicable Credit Course
Foothill GE:	Non-GE
Transferable:	CSU/UC
Grade Type:	Letter Grade (Request for Pass/No Pass)
Repeatability:	Not Repeatable

Student Learning Outcomes

- A successful student will be able to write and debug computer programs which make use of the fundamental control structures and method-building techniques common to all programming languages. Specifically, the student will use data types, input, output, iterative, conditional, and functional components of the language in his or her programs.
- A successful student will be able to use object-oriented programming techniques to design and implement a clear, well-structured computer program. Specifically, the student will use and design classes and objects in his or her programs.

Description

Introduction to basic computer programming concepts using an object-oriented language. Intended for students interested in C S 1A or 2A, but would like a more gradual entry to computing foundations. Coding topics include hands-on practice with software engineering tools, simple programs, variables, control structures, functions, and input/output. Concept topics include the comprehension of specifications, adherence to style guidelines, and the importance of testing to ensure that programs are usable, robust and modifiable.

Course Objectives

The student will be able to:
A. Demonstrate how to use an Integrated Development Environment (IDE) to write a program.
B. Write well documented code in a clear, industry-accepted style.
C. Choose an appropriate data type in which to store a program's data.
D. Convert an English description of a numeric calculation into an expression the computer can evaluate correctly.
E. Incorporate user input into a program to interact with the user.
F. Use appropriate control structures to execute instructions in different sequences.
G. Write a reusable function that solves a common problem.
H. Write code that uses an existing Application Programming Interface (API) to solve a specific problem.
I. Interpret the specifications for, and design and implement solutions to, problems from different application areas.

Course Content

A. Tools
1. Writing vs. running a program
2. Use of command, control, and option keys
3. Navigation through the operating system file structure through well-organized storage and retrieval of files
4. Storage and retrieval of files to/from a server
5. Use compiler, editor and IDE
6. Interpretation of APIs
B. Methodology and Style
1. Test-driven and iterative development methods
2. Compiler errors vs. logic errors
3. Separation of data and computation
4. Documentation
5. Acceptable indentation options
6. Standards and conventions
C. Data Types
1. Primitive data
2. Numeric data
3. Character and string data
4. Boolean data
5. Creating named constants
6. Built-in (language-defined) classes as compound types
D. Expressions
1. Evaluating and creating complex arithmetic expressions
2. Concatenation and string expressions
3. Logical expressions and Boolean algebra
E. Input/Output
1. Unformatted output
2. Simple formatted output
3. User input
a. String data
b. Receiving numeric data directly
c. Receiving string data and converting to a numeric type
F. Basic Control Structures
1. If
2. While
3. Statement block
G. Programmer-defined Functions
1. Parameters
2. Local variables
3. Returned value
H. Predefined Application Programming Interface
1. Parameters
2. Local variables
3. Returned value
I. Applications used throughout Course in Selected Areas
1. Math
2. Physics
3. Chemistry
4. Biology
5. Astronomy
6. Business and finance
7. Internet

Lab Content

A. Using an IDE to write source code for a project and run it
1. Distinguish source code from a recording of the run of a program

2. Include both the source code and a recording of the run in an electronic file(s) for submission
3. Identify a program's errors as originating in the compiler, the program logic, the user's runtime behavior, or the organization of the project in the IDE
B. Using iterative development to progressively refine a project's features to fit a specification
1. Write and test a program that implements just one of a project's required features
2. Add the implementation of a second required feature to the project and test thoroughly
3. Complete the project by implementing and testing the remaining features one by one
4. Perform regression testing after the implementation of each new feature
C. Using Test-driven development to speed up debugging
1. Write test code first that does not run
2. Implement the code required to make the test code run successfully
D. Developing programs that are well designed and easy to modify
1. Outline a project first in English in an abstract way, and make this outline the project's documentation
2. Separate data and computation in a program
3. Use named constants to keep numbers out of a program
4. Choose an appropriate data type for a program's storage
5. Use a consistent and standard indentation style in the source code
E. Writing expressions to be evaluated by the computer
1. Correctly translate an English description of a numeric calculation into an expression that the computer can evaluate
2. Get data from the user in whole numbers and convert so that the calculation takes place with floating point operations and results
3. Write a complex boolean expression
4. Use string manipulations to achieve a specified result
F. Writing a program that interacts with the user
1. Accept character data at runtime from the user to fill a program's variables with values
2. Accept numerical data from the user at runtime to use in calculations
G. Controlling the order in which program statements are executed
1. Use branches
2. Use loops
3. Use function or method calls and returns
4. Enclose groups of statements into blocks to achieve a desired execution sequence
H. Using functions or methods to write code without repetition
1. Write a function or method with no parameters and no returned value
2. Write a function or method with both parameters and a returned value
3. Use the scope of variables to keep data as locally as possible
4. Read an API to find the information needed to effectively call a function or method documented there

Special Facilities and/or Equipment

A. Access to a computer laboratory with the appropriate compilers.
B. Website or course management system with an assignment posting component (through which all lab assignments are to be submitted) and a forum component (where students can discuss course material and receive help from the instructor). This applies to all sections, including on-campus (i.e., face-to-face) offerings.
C. When taught via Foothill Global Access on the internet, the college will provide a fully functional and maintained course management system through which the instructor and students can interact.
D. When taught via Foothill Global Access on the internet, students must have currently existing email accounts and ongoing access to computers with internet capabilities.

Method(s) of Evaluation

Tests and quizzes
Written laboratory assignments, which include source code, sample runs and documentation
Final examination

Method(s) of Instruction

Lectures which include motivation for syntax and use of the object-oriented language, APIs, functional programming, example programs, and analysis of these programs
Online labs (for all sections, including those meeting face-to-face/on campus) consisting of:
1. A programming assignment webpage located on a college-hosted course management system or other department-approved internet environment. Here, the students will review the specification of each programming assignment and submit their completed lab work
2. A discussion webpage located on a college-hosted course management system or other department-approved internet environment. Here, students can request assistance from the instructor and interact publicly with other class members
Detailed review of programming assignments, which includes model solutions and specific comments on the student submissions
In-person or online discussion which engages students and instructor in an ongoing dialog pertaining to all aspects of designing, implementing and analyzing programs
When course is taught fully online:
1. Instructor-authored lecture materials, handouts, syllabus, assignments, tests, and other relevant course material will be delivered through a college-hosted course management system or other department-approved internet environment
2. Additional instructional guidelines for this course are listed in the attached addendum of CS department online practices

Representative Text(s) and Other Materials

Downey and Mayfield. Think Java: How to Think Like a Computer Scientist, 2nd ed.. 2019.

Horstmann and Necaise. Python for Everyon, 2nd ed.. 2016.

Sebesta, Robert. Concepts of Programming Languages, 11th ed.. 2019.

Types and/or Examples of Required Reading, Writing, and Outside of Class Assignments

A. Reading Assignments:
1. Textbook assigned reading averaging 20 pages per week.
2. Reading the supplied handouts and modules averaging 10 pages per week.
3. Reading online resources as directed by instructor though links pertinent to programming.
4. Reading library and reference material directed by instructor through course handouts.
B. Writing Assignments:

1. Writing technical prose documentation that supports and describes the programs that are submitted for grades.

Discipline(s)

Computer Science