Unit 2: Basic Programming Analysis

Content Area: **Computer Science**
Course(s):
Time Period: **Marking Period 1**
Length: **3-4 Weeks**
Status: **Published**

Summary

Students will learn to design a piece of software, or program, and to solve a given problem. They will learn about current programming software functionality. Students will learn about different levels of the debugging process, and how to test their code for all possibilities of input.

Revised Date: July 2025

1.2.12prof.Cr	Creating
L.SS.11–12.1	Demonstrate command of the system and structure of the English language when writing or speaking.
9.4.12.CI.1	Demonstrate the ability to reflect, analyze, and use creative skills and ideas (e.g., 1.1.12prof.CR3a).
P.5	Utilize critical thinking to make sense of problems and persevere in solving them.
9.4.12.CI.2	Identify career pathways that highlight personal talents, skills, and abilities (e.g., 1.4.12prof.CR2b, 2.2.12.LF.8).
L.KL.11–12.2	Apply knowledge of language to understand how language functions in different contexts, to make effective choices for meaning or style, and to comprehend more fully when reading or listening.
P.8	Use technology to enhance productivity increase collaboration and communicate effectively.
9.4.12.CT.2	Explain the potential benefits of collaborating to enhance critical thinking and problem solving (e.g., 1.3E.12profCR3.a).
1.2.12prof.Cr2	Organizing and developing ideas.
1.2.12prof.Cr3	Refining and completing products.
8.1.12.DA.3	Translate between decimal numbers and binary numbers.
8.1.12.DA.4	Explain the relationship between binary numbers and the storage and use of data in a computing device.
8.1.12.AP.3	Select and combine control structures for a specific application based upon performance and readability, and identify trade-offs to justify the choice.
8.1.12.AP.7	Collaboratively design and develop programs and artifacts for broad audiences by incorporating feedback from users.
8.1.12.AP.8	Evaluate and refine computational artifacts to make them more usable and accessible.

Essential Questions / Enduring Understandings

Essential Questions:

- How is object-oriented design implemented into programs?
- Given a problem, what operations need to be performed in order to solve the problem?
- How can you analyze a program's correctness?
- How is an algorithm used in a program?
- How do we write programs in our current programming environment?
- How do we manipulate objects in our current programming environment?
- What are objects?
- How is information encapsulated?
- Given a program (or module of a program), what are the boundary cases?
- Given a program (or module of a program), how would you generate appropriate test data?
- What are good programming practices?

Enduring Understandings:

- Common coding practices are in place to keep standards within the code.
- Reusing code, either in whole or in part, is a positive coding practice.
- Objects can be manipulated in the current programming environment.

Objectives

Students will know:

- ways of adapting a program to changing circumstances
- how to write specifications for software
- how to make adaptable software
- object-oriented development
- encapsulation
- how to choose test data
- how to debug a program

Students will be skilled at:

- evaluating efficiency of a program
- developing programs to scale
- flexibility in programming

Learning Plan

- Preview the essential questions and connect learning throughout the unit.
- Using algorithms, have students develop programs that meet certain specifications.
- Discuss good techniques in designing a user interface.
- Have students develop programs that include a user interface that is appropriate for its given specifications.
- Discuss with the students the purpose of testing modules of their programs.
- Go over the different methods of testing modules depending upon the situation.
- Using programs, discuss which method of testing would be best.
- Have students write programs to a given set of specifications, and then implement a variety of tests upon the program.
- Identify boundary cases for programs and how best to test those boundary cases.
- Expand on existing programs to enable programs to take on larger inputs.

Assessments

- Assessments
    - Formative: Daily assessments using examples from class notes and CodeHS.com, AP Classroom/Albert Checks for Understanding
    - Summative: Teacher-created assessments/projects and CodeHS Computer Science Projects, AP Classroom/Albert Unit Assessments
    - Benchmark: Check for understanding benchmark assessments on CodeHS, AP Classroom/Albert/Khan Academy Diagnostics
    - Alternative Assessments: Student-centered activities such as a doorbell coding project, game design projects, and other activities involving real world applications
        - Complete quizzes/test: Algorithms, Structure of Programs, Design of Programs
        - Be observed by the teacher during individual work on the performance tasks.
        - Conduct self-assessments and reflections
        - Conduct Peer Evaluations.

Materials

- Core instructional materials: Core Book List

- Supplemental materials: CodeHS, computers, and reference books.

Integrated Accommodations and Modifications

See Linked Document.