Greenwich Public Schools Curriculum Overview

Advanced Placement Computer Science A

Personalized learning is achieved through standards-based, rigorous and relevant curriculum that is aligned to digital tools and resources.

Note: Teachers retain professional discretion in how the learning is presented based on the needs and interests of their students.

Course Description

Advanced Placement Computer Science A
Full Year
029200 6 Blocks 1 Credit
Prerequisite: Computer Science Principles or AP Computer Science Principles or Honors Computer Programming for Applications II or Computer Science teacher recommendation.
This course is an Advanced Placement Computer Science curriculum. The course uses the object-oriented computer language Java. Emphasis is on program methodology, algorithms, data structures, advanced data structures and their object-oriented applications. Students taking this course are expected to take the corresponding national Advanced Placement exam given in May.

Unit Guide

Unit 1: Primitive Types
Unit 2: Using Objects
Unit 3: Boolean Expressions and if Statements
Unit 4: Iteration
Unit 5: Writing Classes
Unit 6: Array
Midterm Exam Review & Midterm Exam
Unit 7: ArrayList
Unit 8: 2D Array
Unit 9: Inheritance
Unit 10: Recursion
AP Exam Review
AP Exam / Exit Exam
Post-AP Exam Exploration: TI Innovator Hub, Creating Applets, Graphical Interfaces or other related topics

Computational Thinking Practices

- Program Design and Algorithm Development
- Code Logic
- Code Implementation
- Code Testing
- Documentation

Mathematical Practices

- Make sense of problems and persevere in solving them.
- Reason abstractly and quantitatively.
- Construct viable arguments and critique the reasoning of others.
- Use appropriate tools strategically.
- Attend to precision.
- Look for and make use of structure.
- Look for and express regularity in repeated reasoning.

Enduring Understandings

Units 1 and 2:

- Some objects or concepts are so frequently represented that programmers can draw upon existing code that has already been tested, enabling them to write solutions more quickly and with a greater degree of confidence.
- To find specific solutions to generalizable problems, programmers include variables in their code so that the same algorithm runs using different input values.
- The way variables and operators are sequenced and combined in an expression determines the computed result.

Unit 3:

- The way variables and operators are sequenced and combined in an expression determines the computed result.
- Programmers incorporate iteration and selection into code as a way of providing instructions for the computer to process each of the many possible input values.

Unit 4:

- Programmers incorporate iteration and selection into code as a way of providing instructions for the computer to process each of the many possible input values.

Unit 5:

- Programmers use code to represent a physical object or nonphysical concept, real or imagined, by defining a class based on the attributes and/or behaviors of the object or concept.
- When multiple classes contain common attributes and behaviors, programmers create a new class containing the shared attributes and behaviors forming a hierarchy. Modifications made at the highest level of the hierarchy apply to the subclasses.
- To find specific solutions to generalizable problems, programmers include variables in their code so that the same algorithm runs using different input values.
- While programs are typically designed to achieve a specific purpose, they may have unintended consequences.

Unit 6:

- To manage large amounts of data or complex relationships in data, programmers write code that groups the data together into a single data structure without creating individual variables for each value.
- Programmers incorporate iteration and selection into code as a way of providing instructions for the computer to process each of the many possible input values.

Unit 7:

- To manage large amounts of data or complex relationships in data, programmers write code that groups the data together into a single data structure without creating individual variables for each value.
- Programmers incorporate iteration and selection into code as a way of providing instructions for the computer to process each of the many possible input values.

- While programs are typically designed to achieve a specific purpose, they may have unintended consequences.

Unit 8:

- To manage large amounts of data or complex relationships in data, programmers write code that groups the data together into a single data structure without creating individual variables for each value.
- Programmers incorporate iteration and selection into code as a way of providing instructions for the computer to process each of the many possible input values.

Unit 9:

- When multiple classes contain common attributes and behaviors, programmers create a new class containing the shared attributes and behaviors forming a hierarchy. Modifications made at the highest level of the hierarchy apply to the subclasses.

Unit 10:

- Programmers incorporate iteration and selection into code as a way of providing instructions for the computer to process each of the many possible input values.

Essential Questions

Unit 1:

- How can we use programs to solve problems?
- In what ways are numbers used in the programs and apps you use most often?
- How are mathematical concepts being used in the programs and apps that you use most often?

Unit 2:

- How can we simulate election results using existing program code?
- How are appropriate variables chosen to represent a remote control?
- How do the games we play simulate randomness?

Unit 3:

- How can you use different conditional statements to write a pick-your-own-path interactive story?
- Why is selection a necessary part of programming languages?

Unit 4:

- How does iteration improve programs and reduce the amount of program code necessary to complete a task?
- What situations would warrant the use of one type of loop over another?

Unit 5:

- How can using a model of traffic patterns improve travel time?
- How can programs be written to protect your bank account balance from being inadvertently changed?
- What responsibility do programmers have for the consequences of programs they create, whether intentional or not?

Unit 6:

- How can programs leverage volcano data to make predictions about the date of the next eruption?
- How can knowing standard algorithms be useful when solving new problems?

Unit 7:

- Why is an ArrayList more appropriate for storing your music playlist, while an array might be more appropriate for storing your class schedule?
- How can we use statement execution counts to choose appropriate algorithms?
- What personal data is currently being collected, and how?

Unit 8:

- Why might you want to use a 2D array to store the spaces on a game board or the pixels in a picture, rather than a 1D array or ArrayList?

- Why does the order in which elements are accessed in 2D array traversal matter in some situations?

Unit 9:

- How might the use of inheritance help in writing a program that simulates crops being grown in a virtual world?
- How does inheritance make programs more versatile?

Unit 10:

- What real-world processes do you follow that are recursive in nature?
- Why do programmers sometimes prefer using recursive solutions when sorting data in a large data set?

Resources and Assured Experiences

Textbook Information:

- Fundamentals of Java, Lambert & Osbourne, 2010, ISBN-13: 978-0-538-74492-8
- CSAwesome, Runestone Academy

AP Classroom
Problets
American Computer Science League
Eclipse IDE
TI nSpire and Innovator Hub
GHS Capstone Task, assessed quarterly

- Capstone Vision of the Graduate Capacity: #6 - Generates innovative, creative ideas and products

Quarterly Grading - Quarter Grades will be determined using the following components:

- Participation (includes Classwork) = 5%
- Preparation (includes Homework) = 5%
- Assessments (both Summative & Formative) = 90%

CSTA Computer Standards:

3A-AP-17 Decompose problems into smaller components through systematic analysis, using constructs such as procedures, modules, and/or objects.

3A-AP-19 Systematically design and develop programs for broad audiences by incorporating feedback from users.

3A-AP-21 Evaluate and refine computational artifacts to make them more usable and accessible.

3B-AP-10 Use and adapt classic algorithms to solve computational problems.

3B-AP-12 Compare and contrast fundamental data structures and their uses.

3B-AP-13 Illustrate the flow of execution of a recursive algorithm.

3B-AP-14 Construct solutions to problems using student-created components, such as procedures, modules and/or objects.

3B-AP-20 Use version control systems, integrated development environments (IDEs), and collaborative tools and practices (code documentation) in a group software project.

3B-AP-21 Develop and use a series of test cases to verify that a program performs according to its design specifications.

3B-AP-22 Modify an existing program to add additional functionality and discuss intended and unintended implications (e.g., breaking other functionality).