2- Algorithms and Programming

Content Area:	**Technology**
Course(s):	
Time Period:	**Marking Period 1**
Length:	**17 blocks**
Status:	**Published**

Course Description & Instructional Notes

Course Description

Exam Weighting: 30-35%

All programming languages, whether block-based or text-based, use similar programming structures and commands. Having a basic understanding of how these building blocks are combined to form algorithms and abstractions in one language makes it easier to apply these same understandings to other programming languages. This big idea focuses on determining the efficiency of algorithms, as well as writing and implementing algorithms in a program. This big idea can be paired with any of the other big ideas and taught throughout the school year.

Prior Knowledge
none

Instructional Notes

Building Computational Thinking Practices:
Some problems are so simple that writing a program would be more time-consuming than accomplishing the task by hand, while others are so complicated that it would take a computer an unreasonable amount of time to solve, if it even is possible. It is important for students to understand that not all problems can or should be solved using a program. In procedural abstraction, programmers group program code into a procedure to make the code more readable and reusable. This type of abstraction often means using parameters to make the procedure more general and allow it to be called multiple times with varying inputs. One way to help students create abstractions is to have them write multiple versions of a procedure that each accomplish a specific instance of a task. After identifying the commonalities, students may find it easier to write a more abstract, general procedure that uses parameters to accommodate the differences. In data abstraction, programmers use a list as a representation of something else, such as a grocery list or seating chart. When students explain how their abstractions manage complexity, they should use the context of their specific program to support why the abstraction is necessary or helpful, rather than writing in general about how abstraction manages program complexity. One way to do this is to consider how the program would be written differently, or if it could be written at all, without the use of the abstraction.

Preparing for the AP Exam:
On the AP Exam, students will be asked to determine the result or functionality of code. Students can practice analyzing code segments by first predicting the result of provided code and then checking their hypothesis by running the program on a computer. Because the Create performance task requires students to create their own program, they will need scaffolded practice that moves them from representing algorithmic processes through diagrams or pseudocode, to writing simple programs that perform calculations, to more complex programs that leverage abstraction. While students can collaborate on the development of the program, each student should be a contributing member of the program development. When completing the written responses, which must be completed independently, an in-depth understanding of how the code segments function will allow each

student to answer the prompts on their own.

Technology Integration
Computer Science naturally integrates technology on a daily basis.

Enduring Understandings

To find specific solutions to generalizable problems, programmers represent and organize data in multiple ways.

The way statements are sequenced and combined in a program determines the computed result. Programs incorporate iteration and selection constructs to represent repetition and make decisions to handle varied input values.

Programmers break down problems into smaller and more manageable pieces.By creating procedures and leveraging parameters, programmers generalize processes that can be reused. Procedures allow programmers to draw upon existing code that has already been tested, allowing them to write programs more quickly and with more confidence.

There exist problems that computers cannot solve, and even when a computer can solve a problem, it may not be able to do so in a reasonable amount of time.

Essential Questions

How can we store data in a program to solve problems?

What might happen if you completed the steps in your regular morning routine to get ready and go to school in a different order? How might the reordering affect the decisions you make each morning?

How do video games group the different actions for a player based on what key is pressed on the keyboard or controller? How do apps group different actions together based on user interaction, such as pressing buttons?What types of problems can be solved more easily with a computer, and what types can be solved more easily without a computer? Why?

Student Learning Objectives

Students will be able to…

- Represent a value with a variable.
- Determine the value of a variable as a result of an assignment.
- Represent a list or string using a variable.
- For data abstraction: Develop data abstraction using lists to store multiple elements and explain how the use of data abstraction manages complexity in program code.
- Express an algorithm that uses sequencing without using a programming language.
- Represent a step-by-step algorithmic process using sequential code statements.
- Evaluate expressions that use arithmetic operators.
- Evaluate expressions that manipulate strings.
- For relationships between two variables, expressions, or values: Write expressions using relational operators and evaluate expressions that use relational operators.
- For relationships between Boolean values: Write expressions using logical operators and evaluate expressions that use logic operators.
- Express an algorithm that uses selection without using a programming language.
- For selection: Write conditional statements and determine the result of conditional statements.
- For nested selection: Write nested conditional statements and determine the result of nested conditional statements.
- Express an algorithm that uses iteration without using a programming language.
- For iteration: Write iteration statements and determine the result or side effect of iteration statements.
- Compare multiple algorithms to determine if they yield the same side effect or result.
- For algorithms: Create algorithms and combine and modify existing algorithms.
- For list operations: Write expressions that use list indexing and list procedures and evaluate expressions that use list indexing and list procedures.
- For algorithms involving elements of a list: Write iteration statements to traverse a list and determine the result of an algorithm that includes list traversals.
- For binary search algorithms: Determine the number of iterations required to find a value in a data set and explain the requirements necessary to complete a binary search.
- For procedure calls: Write statements to call procedures and determine the result or effect of a procedure call.
- Explain how the use of procedural abstraction manages complexity in a program.
- Develop procedural abstractions to manage complexity in a program by writing procedures.
- Select appropriate libraries or existing code segments to use in creating new programs.
- For generating random values: Write expressions to generate possible values and evaluate expressions to determine the possible results.
- For simulations: Explain how computers can be used to represent real-world phenomena or outcomes and compare simulations with real-world contexts.
- For determining the efficiency of an algorithm: Explain the difference between algorithms that run in reasonable time and those that do not and identify situations where a heuristic solution may be more appropriate.
- Explain the existence of undecidable problems in computer science.

Vocabulary & Learning Experiences

Essential Academic Vocabulary: list, element, index, string, array, algorithm, sequencing, code statement, expression, modulus, string concatenation, substring, boolean, nested conditional, iteration, traversing

Planned Learning Experiences

Predict and compare

Provide students with a list of expressions with assignments. Ask them to predict the value of each variable after the assignment and then compare their answers to the output produced when these statements are put into a program.

Using manipulatives

When learning about conditionals, take a printout of a simple conditional statement and cut it into multiple sections. As students enter the classroom, hand them an envelope full of the paper strips, and ask them to reassemble the conditional in the proper order.

Marking the text

Provide students with program code that draws a square of side length 10 and a separate set of program code that uses side length 100. Ask students to mark up the sets of code to identify where they are different and to create a generalization by using parameters. Ask them to write a procedure that uses parameters to draw a square of any size.

Think-pair-share

Have students work in pairs to consider what factors would be the most important to prioritize in writing an algorithm to build the perfect master schedule for the school. Some considerations may include maximum class size, student preferences, and teacher availability. Have the pairs discuss and then report their results. Finally, discuss as a class how such programs may have to settle for a “good enough” solution when an exact solution may not be possible in a reasonable amount of time.

Resources

CodeHS
Code.org
MobileCSP

Google Classroom
AP Classroom

Assessments

Formative

Quizzes embedded in CodeHS Modules and Code Review

Better Password Prompt

Students write a program that uses a while loop to prompt a user for a password. They keep prompting the user for the password, and if they get it correct, they then break out of the loop. If they don't get it correct, they should give the user an error message. This activity requires that students use multiple program statements in a specific order to solve a problem.

Summative

Algorithms and Programming Unit Assessment on CodeHS

Practice PT: Tell a Story!

In this activity, students write a JavaScript program that tells a graphical story in at least 4 scenes. Following the milestones and the pseudocode plan that students have laid out prior to this exercise, students write the code for their final project. They iterate and test their code along the way to make sure they have solved each milestone.

Students then reflect upon and answer the following questions:

1. Identify the programming language and purpose of your program.

2. Describe the incremental and iterative development process of your program. How did you divide the program into smaller tasks and make a plan to complete them all?

3. Describe the difficulties and/or opportunities you encountered and how they were resolved or incorporated.

4. Identify an algorithm that is fundamental for your program to achieve its intended purpose and includes two or more additional algorithms.

5. Describe how each algorithm within your selected algorithm functions independently, as well as in combination with others, to form a new algorithm that helps to achieve the intended

purpose of the program.

6. Identify an abstraction you developed, and explain how your abstraction helped manage the complexity of your program.

NJSLS Standards

NJSLS Standards Copied and Pasted as well as linked.

NJSLS Computer Science and Design Thinking

8.2.12.ED.1: Use research to design and create a product or system that addresses a problem and make modifications based on input from potential consumers.
8.2.12.ED.4: Design a product or system that addresses a global problem and document decisions made based on research, constraints, trade-offs, and aesthetic and ethical considerations and share this information with an appropriate audience.
8.2.12.NT.1: Explain how different groups can contribute to the overall design of a product.
8.2.12.NT.2: Redesign an existing product to improve form or function.

Additional NJSLS Standards

NJSLS Standards Copied and Pasted as well as linked.

Interdisciplinary Connections

NJSLS Career Readiness, Life Literacies, and Key Skills

9.4.12.CI.1: Demonstrate the ability to reflect, analyze, and use creative skills and ideas
9.4.12.CT.1: Identify problem-solving strategies used in the development of an innovative product or practice

NJSLS Companion Standards Grades 9-12 (Reading & Writing in Science & Technical Subjects)

Modifications/Accommodations

GENERAL CONSIDERATIONS FOR DIVERSE LEARNERS

English Language Learners

- Personal glossary
- Text-to-speech
- Extended time
- Simplified / verbal instructions
- Frequent breaks

WIDA Can Do Descriptors for Grade 9-12

WIDA Essential Actions Handbook

FABRIC Paradigm

Wall Township ESL Grading Protocol

*Use WIDA Can Do Descriptors in coordination with Student Language Portraits (SLPs).

Students Receiving Special Education Services

- Small group/One to one
- Additional time
- Review of directions
- Student restates information
- Space for movement or breaks
- Extra visual and verbal cues and prompts
- Preferential seating
- Follow a routine/schedule
- Rest breaks
- Verbal and visual cues regarding directions and staying on task
- Checklists
- Immediate feedback

Students receiving Special Education programming have specific goals and objectives, as well as accommodations and modifications outlined within their Individualized Education Plans (IEP) due to an identified disability and/or diagnosis. In addition to exposure to the general education curriculum, instruction is differentiated based upon the student's needs. The IEP acts as a supplemental curriculum guide inclusive of instructional strategies that support each learner.

Considerations for Special Education Students 6-12

Advanced Learners

- Use of high level academic vocabulary/texts
- Problem-based learning
- Pre assess to condense curriculum
- Interest-based research
- Authentic problem-solving
- Homogeneous grouping opportunities

Knowledge and Skill Standards in Gifted Education for All Teachers

Pre-K-Grade 12 Gifted Programming Standards

Gifted Programming Glossary of Terms

Students with 504 Plan

Teachers are responsible for implementing designated services and strategies identified on a student’s 504 Plan.

National Center on Universal Design for Learning - About UDL

UDL Checklist

UDL Key Terms

At Risk Learners / Differentiation Strategies

Alternative Assessments	Independent Research & Projects	Jigsaw
Choice Boards	Multiple Intelligence Options	Think-Tac-Toe
Games and Tournaments	Project-Based Learning	Cubing Activities
Group Investigations	Varied Supplemental Activities	Exploration by Interest
Learning Contracts	Varied Journal Prompts	Flexible Grouping
Leveled Rubrics	Tiered Activities/Assignments	Goal-Setting with Students
Literature Circles	Tiered Products	Homework Options
Multiple Texts	Graphic Organizers	Open-Ended Activities
Personal Agendas	Choice of Activities	Varied Product Choices
Homogeneous Grouping	Mini-Workshops to Reteach or Extend	Stations/Centers
	Think-Pair-Share by readiness or interest	Work Alone/Together
	Use of Collaboration of Various Activities	