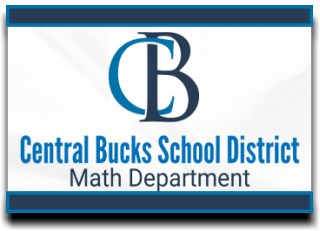

Code and Create

Course Number: 2003

Course Level: Standard/College Bound

Grade: 9-12

Time/Credit: 83 minutes daily: Grades 9, 10, 11, and 12

.5 credits

Development Date: June 2025 – August 2026

Board Approval Date: October 2014

Central Bucks School District
20 Welden Drive
Doylestown, PA 18901

Table of Contents

Central Bucks School District
K-12 Computer Science Teaching and Learning Principles

1. Investigating, problem-solving, and communicating about computer science (speaking, reading, writing, representing, and listening) stimulates students' curiosity, builds confidence, and leads students toward deeper conceptual understanding.

2. Lessons should address students' strengths, interests, and learning styles through differentiated instruction.

3. All students can achieve excellence in computer science when:
    - instruction is appropriately paced
    - teachers set high expectations
    - students feel supported by their teachers, parents and administrators

4. Students must be actively engaged in learning experiences designed to connect to their prior knowledge.

5. For students to transfer and use knowledge meaningfully teachers need to:
    - present new ideas within rich and meaningful contexts
    - help students to make connections to their lives both within and outside of school

6. Computer Science instruction is more effective when teachers incorporate:
    - manipulatives
    - integrated development environments, both online and device based

7. Assessment should be ongoing, diagnostic, and aligned with instruction.

Documents Considered:

College Board (2019). AP Computer Science A: Course and Exam Description. Retrieved from https://apcentral.collegeboard.org/courses.

Computer Science Teachers Association (2017). CSTA K-12 Computer Science Standards, Revised 2017. Retrieved from http://www.csteachers.org/standards.

Kilpatrick, J., Swafford, J., & Findell, B. (2001). *Adding It Up: Helping Children Learn Mathematics.* Washington, DC: National Academy Press.

Marzano, R., & Pickering, D. (1997). *Dimensions of Learning.* Alexandria, VA: Association for Supervision and Curriculum Development; Aurora, CO: Mid-continent Regional Educational Laboratory.

Tomlinson, Carol Ann. (1999). *The Differentiated Classroom: Responding to the Needs of All Learners.* Alexandria, VA: Association for Supervision and Curriculum Development.

Zemelman, S., Daniels, H., & Hyde, A. (1998). *Best practice: New Standards for Teaching and Learning in America's Schools*. Portsmouth, NH: Heinemann.

Central Bucks School District
K-12 Computer Science Program Enduring Understandings

Students will understand that:

Computing Systems

- Abstractions hide the underlying implementation details of computing systems embedded in everyday objects.
- Levels of abstraction and interactions between application software, system software, and hardware layers can be compared.
- Guidelines that convey systematic troubleshooting strategies can be developed that others can use to identify and fix errors.

Networks and the Internet

- There is scalability and reliability of networks, in which the relationship between routers, switches, servers, topology, and addressing can be described.
- Sensitive data can be affected by malware and other attacks.
- Security measures can address various scenarios based on factors such as efficiency, feasibility, and ethical impacts.

Data and Analysis

- There are different bit representations of real-world phenomena, such as characters, numbers, and images.
- There are tradeoffs in how data elements are organized and where data is stored.

Algorithms and Programming

- Algorithms are used to solve computational problems by leveraging prior student knowledge and personal interests.
- Programmers must justify the selection of specific control structures when tradeoffs involve implementation, readability, and program performance, and explain the benefits and drawbacks of choices made.
- Problems can be decomposed into smaller components through systematic analysis, using constructs such as procedures, modules, and/or objects.
- Documentation of design decisions using text, graphics, presentations, and/or demonstrations is an integral part in the development of complex programs.

Impacts of Computing

- Computing affects personal, ethical, social, economic, and cultural practices.
- Computational artifacts must be tested and refined to reduce bias and equity deficits.

Central Bucks School District
K-12 Computer Science Program Essential Questions

Computing Systems

- What is an abstraction?
- How does abstraction manage complexity?

Networks and the Internet

- How do scalability and redundancy relate to reliability on the Internet?
- What is malware and how is sensitive data affected by it?
- How do efficiency, feasibility, and ethical impacts impact security measures?

Data and Analysis

- What are the tradeoffs in how data elements are organized and where data is stored?
- How do we collect, visualize, and store large amounts of data?
- Why do we create computational models and how does the data we use affect those models?
- What is the purpose of creating prototypes to solve computational problems?

Algorithms and Programming

- What is an algorithm?
- How do algorithms and programs differ?
- Why is the selection of specific control structures important when considering implementation, readability, and program performance?
- Why is the decomposition into smaller components in constructs such as procedures, modules, and/or objects important in the software development process?
- Why are documentation and design decisions an integral part in the development of complex programs?

Impacts of Computing

- What is the purpose of collecting data?
- How is data used or abused?
- What are the ways that computing affects personal, ethical, social, economic, and cultural practices?
- Why must we test and refine computational artifacts to reduce bias and equity deficits?

Central Bucks School District
Code and Create Overview

Desired Results

Course Description: 2003 Code & Create (9 weeks, .5 credits)

Unlock your creativity with Code and Create, a dynamic course designed for high school students interested in block-based coding. Using an object-oriented programming language, students will learn to program virtual worlds in a 3D environment. This course emphasizes algorithmic thinking, abstraction, and problem-solving skills. Students will create movie-style animations, interactive games, and simulations, bringing their ideas to life. Up to one credit from Computer Science courses may be used to fulfill graduation requirements for mathematics.

Prerequisite: None

Committee Members**:** Dustin Frey, Aaron Fromal, Jason Matusek, Anthony Paglione, Jenn Robinson, Brian Novick, Math Supervisor

Acceptable Evidence

Final Project: In lieu of a final written exam, final project(s) will be required. The final project grade is 20% of the final course grade.

Code and Create Pacing Guide

Instructional Timeline	Blocks
Unit 1: Getting to Know your IDE	**8**
• Basics of your IDE (i.e. Alice, Snap, Scratch, etc.) • Design Process (Scenarios, storyboards, etc.) • Built-in Methods, Functions, and Expressions • Simple Control Structures • A First Program (i.e. a guided project)	7
Unit 1 Summative Assessment	1
Unit 2: Programming Basics	**8**
• Algorithms • Primitive Methods/Procedures/Functions • Properties • Parameters • Debugging • Comments • Project (Reteaching and Enrichment as necessary)	7
Unit 2 Summative Assessment	1
Unit 3: Control Structures: Loops & Conditionals	**8**
• Execution Control with If/Else and Boolean Functions • Operators (relational/logical) • For Loops • While Loops • Project (Reteaching and Enrichment as necessary)	7
Unit 3 Summative Assessment	1
Unit 4: Smart Structures: Methods & Reuse	**8**
• User-Defined Methods • Abstraction • Modularity • Re-use • Project (Reteaching and Enrichment as necessary)	7
Unit 4 Summative Assessment	1
Unit 5: Events and Handling	**8**
• Interactive Programming • Parameters and Event Handling Methods • Project (Reteaching and Enrichment as necessary)	7
Unit 5 Summative Assessment	1
Final Project	**5**
Total Days for Course:	**45**

Required and Supplementary Texts/Software

Required Texts/Software

None. This course is taught with online materials. These sites provide free resources to schools to promote quality computer science instruction in K–12 education.

Suggested programming environments include, but are not limited to:

Alice 3.0 (preferred) https://www.alice.org/get-alice/alice-3/ (through Carnegie Mellon University)
Scratch - http://scratch.mit.edu - Scratch - Imagine, Program, Share (through MIT)
Snap - https://snap.berkeley.edu Snap! 11.0.8 (through Berkeley University)
MakeCode Arcade (free through Microsoft) Microsoft MakeCode Arcade
www.Code.org (game lab)
www.codehs.com

Technology

Computers: Student 1:1 Devices

Central Bucks School District – Course of Study

Unit 1: Getting to Know your IDE

Desired Results

Essential Questions:

- How does an Integrated Development Environment (IDE) help programmers create, test, and refine their programs?
- What features of our IDE make programming more visual or beginner-friendly?
- Why is understanding your programming environment an important first step for learning to code?
- How can careful planning improve the quality and efficiency of a program?
- What role do storyboards or scenarios play in transforming an idea into a working program?
- How does the design process in programming compare to design processes in other creative fields?
- How do built-in functions and methods simplify the process of programming?
- What's the difference between a method, a function, and an expression—and why does it matter?
- How can we use existing programming tools effectively instead of creating everything from scratch?
- How do control structures (such as sequence, selection, and repetition) allow a program to make decisions and repeat actions?
- Why are control structures essential for creating dynamic and interactive programs?
- How can we trace or predict what a program will do based on its control flow?
- What steps are needed to turn a concept or design into a functional program?
- How do debugging and testing help ensure that a program behaves as intended?
- What did creating your first program teach you about how computers follow directions?

Enduring Understandings:

Students will understand that...

- They can navigate and use key features of an Integrated Development Environment (IDE) to create, edit, and run programs.
- The IDE's tools support problem-solving, creativity, and debugging in programming.
- Basic programming actions, such as adding objects, commands, or blocks, are used to build simple projects.
- The design process helps plan and organize a program before writing code.
- Storyboards and/or flowcharts can represent a program's logic and sequence.
- Designs can be refined and improved through feedback, testing, and reflection.
- Built-in methods and functions can perform common programming tasks efficiently.
- Expressions combine values, variables, and operations to produce results.
- Using appropriate built-in tools can simplify coding and reduce errors.
- Control structures (e.g., sequence, selection, and repetition) determine how a program flows.
- Control structures affect the behavior and outcome of a program.
- Logic and reasoning are essential for tracing and debugging simple control structures.
- Applying the full design and programming process leads to the creation of a simple, functional program.
- Testing, debugging, and revising are necessary to ensure a program meets design goals and user expectations.

- Reflecting on problem-solving strategies helps improve future programming projects.

Knowledge:
Students will know…

- Definitions: IDE, interface, script, code, block, command, run, execute, debug, debugging, syntax, output, design process, storyboard, flowchart, scenario, algorithm, iteration, method, function, parameter, expression, operator, variable, sequence, selection, conditional, loop, repetition, Boolean, condition, event, program, testing, bug, algorithmic thinking, documentation

Skills:
Students will be able to…

- Navigate and use features of an IDE (Alice, Snap, Scratch, etc.)
- Create, edit, and run simple programs
- Apply the design process to plan and organize a project
- Develop and interpret storyboards and flowcharts
- Write and follow basic algorithms
- Use built-in methods and functions effectively
- Utilize expressions, variables, and operators to produce results
- Apply simple control structures (sequence, selection, and loops)
- Trace and debug errors in a program
- Use events and conditions to make programs interactive
- Test and refine programs based on feedback and outcomes
- Document code and explain program logic
- Collaborate and share ideas in the design and coding process
- Reflect on how problem-solving and creativity apply to programming

Standards:
Algorithms and Programming

Priority Standards

3A-AP-13	Create prototypes that use algorithms to solve computational problems by leveraging prior student knowledge and personal interests.
3A-AP-16	Design and iteratively develop computational artifacts for practical intent, personal expression or to address a societal issue by using events to initiate instructions.
3A-AP-18	Create artifacts by using procedures within a program, combinations of data and procedures, or independent but interrelated programs.
3B-AP-14	Construct solutions to problems using student created components, such as procedures, modules and/or objects.

Supporting Standards

3A-AP-17	Decompose problems into smaller components through systematic analysis, using constructs such as procedures, modules, and/or objects.
3B-AP-13	Illustrate the flow of execution of recursive algorithms.

Acceptable Evidence

Acceptable evidence of learning other than Performance Tasks:

Formative Assessment

- Warm-ups
- Checkpoint assignments
- Classwork
- Homework
- Exit slips
- Keep a notebook
- Programming assignments

Summative Assessment

- Unit Quiz
- Unit Project

Suggested learning experiences and instruction

Learning Experiences: Will vary based on the IDE used for the course.

Instructional Strategies:

- Demonstrate writing programs in the selected IDE.
- Use PowerPoint slides to present concepts and terminology.
- Emphasize the steps in problem solving and the use of algorithms.

Differentiation:

- Work with students in small groups to intervene or enrich as needed
- Provide additional scaffolds for emerging learners.
- Allow on level learners to modify existing code and explain their changes.
- Provide opportunities for advanced learners to add features, variables, or additional functionality.

For Language Development:

- Create a student-made dictionary that includes terms, definitions, and examples organized by chapter.
- Make flash cards for terms and examples, then mix and match.
- Create study guides with clear definitions and vocabulary development opportunities.

For Advanced Learners:

- Meet in small groups with students who need enrichment to provide instruction and learning activities on the optional skills followed up by an alternative and more complex assignment.
- Excuse the students from standard practice and use appropriately challenging practice that requires students to apply the concepts in more rigorous ways to challenge their thinking and allow students to go deeper on the same skills.
- Provide more rigorous opportunities for application of skills including enrichment projects.

Central Bucks School District – Course of Study

Unit 2: Programming Basics

Desired Results

Essential Questions:

- How do algorithms help us break down and solve problems step by step?
- What makes one algorithm more efficient or effective than another?
- How can we translate real-world processes into algorithms that a computer can understand?
- How do methods, procedures, and functions help organize and simplify code?
- Why is it important to break a large program into smaller, reusable pieces?
- How do we decide when to create a new function versus using a built-in one?
- How do object properties affect how a program looks and behaves?
- What is the relationship between an object's properties and the code that controls them?
- How can changing a property help a program achieve a desired result?
- How do parameters make functions and methods more flexible and reusable?
- What happens when we pass different parameters to the same function or method?
- How do programmers find and fix errors in their code?
- What strategies or tools can make debugging more efficient?
- Why is debugging an essential part of the problem-solving process?
- How do programmers collaborate with others to debug programs?
- How do comments make code easier to understand, share, and maintain?
- Why is writing clear comments as important as writing functional code?
- How can comments be used to plan, explain, or document a program's logic?
- How can we apply what we've learned about algorithms, functions, and debugging to create an effective program?
- What steps should we take to evaluate and improve our own or others' code?
- How does the programming process reflect persistence, creativity, and attention to detail?

Enduring Understandings:

Students will understand that…

- Algorithms provide a clear, logical sequence of steps to solve problems effectively.
- Different algorithms can solve the same problem, but some are more efficient or easier to understand.
- Methods, procedures, and functions help organize programs into smaller, reusable parts that make code easier to manage.
- Properties define the characteristics and behavior of objects within a program.
- Changing object properties can control how a program looks, feels, and responds to user input.
- Parameters allow functions and methods to work flexibly with different inputs and situations.
- Debugging is an essential skill that requires patience, logic, and attention to detail.
- Errors in programming are opportunities to learn and improve both the code and the problem-solving process.
- Comments make code easier for others (and the programmer) to read, understand, and maintain over time.
- Writing clear, well-organized, and well-documented code is a mark of good programming practice.

- Effective programming involves designing, testing, and refining solutions through continuous improvement.
- Collaboration, communication, and reflection are key to improving both individual and group programming projects.

Knowledge:
Students will know…

- Definitions: algorithm, method, procedure, function, property, parameter, variable, value, argument, object, debugging, error, bug, syntax, runtime, comment, documentation, input, output, sequence, selection, repetition, conditional, loop, logic, trace, test, program, code, IDE, project, iteration

Skills:
Students will be able to…

- Design and follow algorithms to solve problems step by step
- Write, call, and organize code using methods, procedures, and functions
- Define and modify object properties to control program behavior
- Use parameters to make functions and methods flexible and reusable
- Identify, trace, and fix errors in programs through debugging
- Write clear comments to explain code logic and document programs
- Combine variables, expressions, and operations to produce desired outputs
- Apply sequence, selection, and repetition to control program flow
- Test and refine programs to ensure they meet design requirements
- Reflect on and improve programming approaches based on feedback and results
- Collaborate with others to plan, develop, and troubleshoot code
- Apply computational thinking to plan, analyze, and implement solutions

Standards:

Algorithms and Programming

Priority Standards

3A-AP-13	Create prototypes that use algorithms to solve computational problems by leveraging prior student knowledge and personal interests.
3A-AP-16	Design and iteratively develop computational artifacts for practical intent, personal expression or to address a societal issue by using events to initiate instructions.
3A-AP-18	Create artifacts by using procedures within a program, combinations of data and procedures, or independent but interrelated programs.
3B-AP-14	Construct solutions to problems using student created components, such as procedures, modules and/or objects.

Supporting Standards

3A-AP-15	Justify the selection of specific control structures when tradeoffs involve implementation, readability, and program performance, and explain the benefits and drawbacks of choices made.

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.
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 recursive algorithms.

Acceptable Evidence

Acceptable evidence of learning other than Performance Tasks:

Formative Assessment

- Warm-ups
- Checkpoint assignments
- Classwork
- Homework
- Exit slips
- Keep a notebook
- Programming assignments

Summative Assessment

- Unit Quiz
- Unit Project

Suggested learning experiences and instruction

Learning Experiences: Will vary based on the IDE used for the course.

Instructional Strategies:

- Demonstrate writing programs in the selected IDE.
- Use PowerPoint slides to present concepts and terminology.
- Emphasize the steps in problem solving and the use of algorithms.

Differentiation:

- Work with students in small groups to intervene or enrich as needed
- Provide additional scaffolds for emerging learners.
- Allow on level learners to modify existing code and explain their changes.
- Provide opportunities for advanced learners to add features, variables, or additional functionality.

For Language Development:

- Create a student-made dictionary that includes terms, definitions, and examples organized by chapter.
- Make flash cards for terms and examples, then mix and match.
- Create study guides with clear definitions and vocabulary development opportunities.

For Advanced Learners:

- Meet in small groups with students who need enrichment to provide instruction and learning activities on the optional skills followed up by an alternative and more complex assignment.
- Excuse the students from standard practice and use appropriately challenging practice that requires students to apply the concepts in more rigorous ways to challenge their thinking and allow students to go deeper on the same skills.
- Provide more rigorous opportunities for application of skills including enrichment projects.

Central Bucks School District – Course of Study

Unit 3: Control Structures: Loops & Conditionals

Desired Results

Essential Questions:

- How do if/else statements and Boolean functions allow a program to make decisions?
- How can relational and logical operators be used to test conditions and control program flow?
- What are the differences between for loops and while loops, and when should each be used?
- How can loops help automate repetitive tasks and simplify code?
- How do we combine conditional statements and loops to create more complex and dynamic programs?
- How can testing and debugging strategies ensure that control structures and loops work correctly in a program?

Enduring Understandings:

Students will understand that…

- If/else statements allow programs to make decisions based on conditions.
- Boolean functions evaluate to true or false and are essential for controlling program flow.
- Relational operators compare values, and logical operators combine conditions to guide decisions.
- For loops execute a block of code a predetermined number of times, simplifying repetitive tasks.
- While loops execute code repeatedly until a condition is no longer true, enabling flexible iteration.
- Combining conditional statements and loops allows programs to respond dynamically to different situations.
- Effective use of control structures and loops improves program efficiency, readability, and maintainability.
- Testing, debugging, and refining control structures ensures that programs behave as intended.
- Planning and applying loops and conditionals strategically supports the development of complete programming projects.

Knowledge:

Students will know…

- Definitions: if/else statement, Boolean, Boolean function, relational operator, logical operator, comparison, condition, true, false, for loop, while loop, iteration, loop control, nested loop, counter, accumulator, conditional statement, expression, program flow, control structure, debugging, test, trace, variable, logic, algorithm, project

Skills:

Students will be able to…

- Use if/else statements to control the flow of a program based on conditions
- Evaluate and apply Boolean functions to make decisions in code
- Use relational and logical operators to compare values and combine conditions
- Implement for loops to execute a block of code a predetermined number of times
- Implement while loops to repeat code until a specific condition is met
- Nest loops and conditional statements to solve more complex problems

- Trace the flow of a program to predict its behavior and identify errors
- Debug programs that use loops and conditional statements
- Plan and design solutions that use iteration and decision-making structures
- Optimize code by selecting appropriate loops and conditions for a task
- Apply control structures to complete projects efficiently and correctly
- Test and refine programs to ensure loops and conditionals work as intended

Standards:

Priority Standards

3A-AP-13	Create prototypes that use algorithms to solve computational problems by leveraging prior student knowledge and personal interests.
3A-AP-16	Design and iteratively develop computational artifacts for practical intent, personal expression or to address a societal issue by using events to initiate instructions.
3A-AP-18	Create artifacts by using procedures within a program, combinations of data and procedures, or independent but interrelated programs.
3B-AP-14	Construct solutions to problems using student created components, such as procedures, modules and/or objects.

Supporting Standards

3A-AP-15	Justify the selection of specific control structures when tradeoffs involve implementation, readability, and program performance, and explain the benefits and drawbacks of choices made.
3A-AP-17	Decompose problems into smaller components through systematic analysis, using constructs such as procedures, modules, and/or objects.
3A-AP-21	Evaluate and refine computational artifacts to make them more usable and accessible.
3B-AP-08	Describe how artificial intelligence drives many software and physical systems
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 recursive algorithms.

Acceptable Evidence

Acceptable evidence of learning other than Performance Tasks:

Formative Assessment

- Warm-ups
- Checkpoint assignments
- Classwork
- Exit slips
- Keep a notebook
- Programming assignments

Summative Assessment

- Unit Quiz
- Unit Project

Suggested learning experiences and instruction

Instructional Strategies:

- Demonstrate writing programs in the selected IDE.
- Use PowerPoint slides to present concepts and terminology.
- Emphasize the steps in problem solving and the use of algorithms.

Differentiation:

- Work with students in small groups to intervene or enrich as needed
- Provide additional scaffolds for emerging learners.
- Allow on level learners to modify existing code and explain their changes.
- Provide opportunities for advanced learners to add features, variables, or additional functionality.

For Language Development:

- Create a student-made dictionary that includes terms, definitions, and examples organized by chapter.
- Make flash cards for terms and examples, then mix and match.
- Create study guides with clear definitions and vocabulary development opportunities.

For Advanced Learners:

- Meet in small groups with students who need enrichment to provide instruction and learning activities on the optional skills followed up by an alternative and more complex assignment.
- Excuse the students from standard practice and use appropriately challenging practice that requires students to apply the concepts in more rigorous ways to challenge their thinking and allow students to go deeper on the same skills.
- Provide more rigorous opportunities for application of skills including enrichment projects.

Central Bucks School District – Course of Study

Unit 4: Smart Structures: Methods & Reuse

Desired Results

Essential Questions:

- How do user-defined methods allow programmers to simplify and organize code?
- How does abstraction help focus on essential features while ignoring unnecessary details?
- Why is modularity important for creating large, maintainable programs?
- How can reusing code improve efficiency and reduce errors in programming projects?
- How do user-defined methods, abstraction, and modularity work together to solve complex problems?
- How can testing, debugging, and refining individual modules improve the overall quality of a program?
- How can these concepts be applied to complete programming projects effectively and creatively?

Enduring Understandings:

Students will understand that…

- User-defined methods allow programmers to break down tasks into smaller, manageable pieces.
- Abstraction helps programmers focus on the essential behavior of a program without getting lost in details.
- Modularity organizes code into independent, reusable sections that improve readability and maintainability.
- Reusing methods and code components increases efficiency and reduces the likelihood of errors.
- Combining methods, abstraction, and modularity supports the development of more complex and flexible programs.
- Testing and debugging individual methods or modules ensures the program functions correctly as a whole.
- Applying these principles promotes good programming practices and effective problem-solving in projects.

Knowledge:

Students will know…

- Definitions: user-defined method, method, procedure, function, abstraction, modularity, module, reuse, parameter, argument, return value, variable, object, class, code block, procedure call, program structure, encapsulation, debugging, test, trace, program flow, project, algorithm

Skills:

Students will be able to…

- Create and call user-defined methods to organize code
- Pass parameters and use return values to make methods flexible
- Apply abstraction to focus on essential program features and hide unnecessary details
- Break programs into modular components for readability and maintainability
- Reuse methods and code blocks to improve efficiency and reduce errors
- Combine multiple methods to solve complex problems
- Trace the flow of a program that uses user-defined methods and modular design

- Debug programs that include methods and modular components
- Plan and design programs using modularity and abstraction principles
- Apply abstraction and modularity to complete programming projects successfully
- Evaluate and refine code for reusability, efficiency, and clarity

Standards:

Algorithms and Programming

Priority Standards

3A-AP-13 Create prototypes that use algorithms to solve computational problems by leveraging prior student knowledge and personal interests.

3A-AP-16 Design and iteratively develop computational artifacts for practical intent, personal expression or to address a societal issue by using events to initiate instructions.

3A-AP-18 Create artifacts by using procedures within a program, combinations of data and procedures, or independent but interrelated programs.

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

Supporting Standards

3A-AP-15 Justify the selection of specific control structures when tradeoffs involve implementation, readability, and program performance, and explain the benefits and drawbacks of choices made.

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

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

3B-AP-09 Implement an artificial intelligence algorithm to play a game against a human opponent or solve a problem.

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 recursive algorithms.

Acceptable Evidence

Acceptable evidence of learning other than Performance Tasks:

Formative Assessment

- Warm-ups
- Checkpoint assignments
- Classwork
- Homework
- Exit slips
- Keep a notebook
- Programming assignments

Summative Assessment

- Unit Quiz
- Unit Project

Suggested learning experiences and instruction

Instructional Strategies:

- Demonstrate writing programs in the selected IDE.
- Use PowerPoint slides to present concepts and terminology.
- Emphasize the steps in problem solving and the use of algorithms.

Differentiation:

- Work with students in small groups to intervene or enrich as needed
- Provide additional scaffolds for emerging learners.
- Allow on level learners to modify existing code and explain their changes.
- Provide opportunities for advanced learners to add features, variables, or additional functionality.

For Language Development:

- Create a student-made dictionary that includes terms, definitions, and examples organized by chapter.
- Make flash cards for terms and examples, then mix and match.
- Create study guides with clear definitions and vocabulary development opportunities.

For Advanced Learners:

- Meet in small groups with students who need enrichment to provide instruction and learning activities on the optional skills followed up by an alternative and more complex assignment.
- Excuse the students from standard practice and use appropriately challenging practice that requires students to apply the concepts in more rigorous ways to challenge their thinking and allow students to go deeper on the same skills.
- Provide more rigorous opportunities for application of skills including enrichment projects.

Central Bucks School District – Course of Study

Unit 5: Events & Handling

Desired Results

Essential Questions:

- How do events allow programs to respond to user interactions or other triggers?
- How can parameters be used with event-handling methods to create dynamic responses in a program?
- What are the differences between interactive programming and sequential programming?
- How can event-driven programming make programs more engaging and responsive?
- How do we design, test, and debug event-handling methods to ensure correct program behavior?
- How can events and event-handling methods be combined to create interactive projects efficiently?

Enduring Understandings:

Students will understand that…

- Events enable programs to respond to user actions, system signals, or other triggers.
- Event-handling methods allow programmers to define specific responses to events.
- Parameters passed to event-handling methods provide information needed to respond dynamically.
- Interactive programming makes programs more responsive and engaging for users.
- Combining events and event-handling methods allows for more flexible and modular program design.
- Debugging and testing event-driven programs ensures that all possible interactions are handled correctly.
- Applying event-driven principles supports the creation of complete and interactive programming projects.

Knowledge:

Students will know…

- Definitions: event, event handler, event-driven programming, interactive programming, parameter, argument, method, function, procedure, callback, trigger, user input, mouse click, key press, program response, listener, object, object property, debugging, test, trace, project, program flow, handler method

Skills:

Students will be able to…

- Identify events and define event-handling methods to respond to them
- Use parameters in event-handling methods to create dynamic responses
- Design programs that respond to user interactions and system events
- Implement interactive programming techniques using objects, properties, and event handlers
- Combine multiple event-handling methods to create complex, responsive behaviors
- Trace and debug programs that include interactive events and handlers
- Apply testing strategies to ensure all events are handled correctly in a program

- Plan, design, and complete projects that use interactive programming and event handling
- Evaluate and refine interactive programs for usability, functionality, and responsiveness

Standards:

Algorithms and Programming

Priority Standards

3A-AP-13 Create prototypes that use algorithms to solve computational problems by leveraging prior student knowledge and personal interests.

3A-AP-14 Use lists to simplify solutions, generalizing computational problems instead of repeatedly using simple variables.

3A-AP-16 Design and iteratively develop computational artifacts for practical intent, personal expression or to address a societal issue by using events to initiate instructions.

3A-AP-18 Create artifacts by using procedures within a program, combinations of data and procedures, or independent but interrelated programs.

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

Supporting Standards

3A-AP-15 Justify the selection of specific control structures when tradeoffs involve implementation, readability, and program performance, and explain the benefits and drawbacks of choices made.

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-09 Implement an artificial intelligence algorithm to play a game against a human opponent or solve a problem.

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

3B-AP-11 Evaluate algorithms in terms of their efficiency, correctness, and clarity.

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

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

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

3B-AP-23 Evaluate key qualities of a program through a process such as a code review.

Acceptable Evidence

Acceptable evidence of learning other than Performance Tasks:

Formative Assessment

- Warm-ups
- Checkpoint assignments
- Classwork
- Homework
- Exit slips
- Keep a notebook
- Programming assignments

Summative Assessment

- Unit Quiz
- Unit Project

Suggested learning experiences and instruction

Instructional Strategies:

- Demonstrate writing programs in the selected IDE.
- Use PowerPoint slides to present concepts and terminology.
- Emphasize the steps in problem solving and the use of algorithms.

Differentiation:

- Work with students in small groups to intervene or enrich as needed
- Provide additional scaffolds for emerging learners.
- Allow on level learners to modify existing code and explain their changes.
- Provide opportunities for advanced learners to add features, variables, or additional functionality.

For Language Development:

- Create a student-made dictionary that includes terms, definitions, and examples organized by chapter.
- Make flash cards for terms and examples, then mix and match.
- Create study guides with clear definitions and vocabulary development opportunities.

For Advanced Learners:

- Meet in small groups with students who need enrichment to provide instruction and learning activities on the optional skills followed up by an alternative and more complex assignment.
- Excuse the students from standard practice and use appropriately challenging practice that requires students to apply the concepts in more rigorous ways to challenge their thinking and allow students to go deeper on the same skills.
- Provide more rigorous opportunities for application of skills including enrichment projects.