SOFTWARE DEVELOPMENT, 15.1200.40	
1.0 APPLY PROBLEM-SOLVING AND CRITICAL THINKING SKILLS TO INFORMATION TECHNOLOGY	
1.1	Describe methods and considerations for prioritizing and scheduling software development tasks (e.g., risk-value, waterfall, agile, GTD, Kanban)
1.2	Describe methods and techniques of problem-solving and troubleshooting applicable to software development
2.0 MAINTAIN A SAFE GREEN INFORMATION TECHNOLOGY ENVIRONMENT	
2.1	Identify ergonomics and repetitive strain injuries common in information technology
2.2	Describe programming techniques to analyze and reduce power consumption (e.g., Algorithmic Efficiency, Big O notation)
2.3	Describe the environmental importance of the proper disposal of computer components, including safe disposal of toxic chemicals
2.4	Explain the impact of large scale computing (e.g., power consumption, thermal impact)
3.0 RECOGNIZE GENERAL SECURITY ISSUES RELATED TO INFORMATION TECHNOLOGY	
3.1	Explain procedures to maintain data integrity and security (e.g., lock the screen, don't open unrecognized emails, don't plug in untrusted thumb drives, only use approved software)
3.2	Identify security issues related to the network, computer hardware, software, and data
3.3	Describe computer threats and methods to protect a computer (e.g., viruses, phishing, e-mail, social engineering, spoofing, identify theft, spamming
3.4	Explain concepts such as denial of service, hacking/cracking, intrusion, detection, and prevention
4.0 RECOGNIZE SPECIFIC SECURITY ISSUES RELATED TO SOFTWARE DEVELOPMENT	
4.1	Explain best practices for maintaining integrity and security in software development (e.g., encryption, hashing, digital signatures)
4.2	Describe methods for sanitizing user input to prevent issues such as buffer overflows and SQL injection
4.3	Explain the difference between authentication and authorization
4.4	Explain how software defects relate to software security (e.g., buffer overflows, cross-site scripting)
5.0 EXPLORE LEGAL AND ETHICAL ISSUES RELATED TO INFORMATION TECHNOLOGY	
5.1	Explore intellectual property rights including software licensing and software duplication
5.2	Explain legal and ethical issues related to the difference between various open source and proprietary licenses

	(e.g., GPL, BSD, and proprietary licenses)
5.3	Identify issues and trends affecting computers and information privacy (e.g., HIPAA, COPPA, CISPA, FERPA, data brokers)
5.4	Describe licensing considerations for using third party libraries when creating software for sale (e.g. open-source restrictions, proprietary restrictions, copyleft licenses)
5.5	Differentiate between ethical and legal uses of information technology (e.g., data pricing, use of public and private networks, social networking, industry-related data, data piracy)
6.0 DEMONSTRATE BASIC COMPUTER MATHEMATICS REQUIRED FOR INFORMATION TECHNOLOGY	
6.1	Explain the function of base number systems in mathematics as they relate to computer technology
6.2	Perform decimal to binary and binary to decimal conversions
6.3	Perform decimal to hexadecimal and hexadecimal to decimal conversions
6.4	Perform hexadecimal to binary and binary to hexadecimal conversions
6.5	Determine the appropriate method to perform conversions (e.g., paper/pencil, electronic resources)
6.6	Perform modulo and bitwise operations
7.0 DESCRIBE THE DEVELOPMENT/EVOLUTION OF COMPUTERS AND INFORMATION TECHNOLOGY	
7.1	Describe a computer and its components and functions
7.2	Describe the evolution of the computer through its generations (e.g., vacuum tubes, transistors, integrated circuits)
7.3	Explain the evolution of programming languages from low level to high level (e.g., machine code, assembly, higher-level languages)
7.4	Explain how the development of computers has impacted modern life
7.5	Identify future trends in computing (e.g., quantum, optical, DNA)
8.0 USE VERSION CONTROL SYSTEMS	
8.1	Identify the purpose of version control systems (e.g., Git, Mercurial)
8.2	Create a new repository
8.3	Add source code or data to repository
8.4	Push changes to an upstream/master repository

8.5	Pull changes from the upstream/master repository
8.6	Restore previous versions of code from the repository
9.0 DEMONSTRATE PROGRAM ANALYSIS AND DESIGN	
9.1	Explain the steps in the System Development Life Cycle (SDLC) (e.g., planning, analysis, design, development, testing, implementation, maintenance)
9.2	Interpret a problem statement and identify program requirements
9.3	Develop program requirements/specifications and a testing plan (e.g., user stories, automated testing, test procedures)
9.4	Determine input and output
9.5	Choose appropriate data structures
9.6	Differentiate between bottom-up and top-down design
9.7	Apply pseudocode or graphical representations to plan the structure of a program or module (e.g., flowcharting, white boarding, UML)
9.8	Choose appropriate documentation for a module
9.9	Differentiate among major programming paradigms (e.g., imperative vs. declarative, object-oriented, functional)
10.0 DEVELOP A PROGRAM USING INDUSTRY BEST PRACTICES	
10.1	Use a program editor to enter and modify code
10.2	Differentiate between interpreted and compiled code (e.g., steps necessary to run executable code)
10.3	Identify the purpose of a build system (e.g., make, rake, ant, maven, SCons, grunt)
10.4	Apply industry standards in documentation (e.g., self-documenting code; function-level, program-level, and user-level documentation)
10.5	Name identifiers and formatting code by applying recognized conventions
10.6	Find and use program and language documentation
11.0 TEST AND DEBUG TO VERIFY PROGRAM OPERATION	
11.1	Identify errors in program modules
11.2	Identify boundary cases and generate appropriate test data

11.3	Perform integration testing including tests within a program to protect execution from bad input or other run-time errors
11.4	Categorize, identify, and correct errors in code, including syntax, semantic, logic, and runtime.
11.5	Perform different methods of debugging (e.g., hand-trace code or real time debugging tools)
12.0 WRITE CODE TO PERFORM ARITHMETIC CALCULATIONS	
12.1	Identify and correctly use arithmetic operations, applying the order of operations (precedence) with respect to programming
12.2	Interpret and construct mathematical formulas
12.3	Use basic algorithms
12.4	Identify correct and problematic uses of integers, floating-point numbers, and fixed-point numbers in arithmetic
13.0 EMPLOY MODULARITY IN WRITING PROGRAMS	
13.1	Use standard library functions
13.2	Find and use third party libraries (e.g., on the web, package managers)
13.3	Demonstrate refactoring techniques to reduce repetitious code and improve maintainability
13.4	Demonstrate the use of parameters to pass data into program modules
13.5	Demonstrate the use of return values from modules
14.0 UTILIZE CONDITIONAL STRUCTURES IN WRITING PROGRAMS	
14.1	Select correct syntax for decision statements (e.g., if/else, if, switch case)
14.2	Compare values using relational operators (e.g., =, >, <, >=, <=, not equal)
14.3	Evaluate Boolean expressions (e.g., AND, OR, NOT, NOR, XOR)
14.4	Select an appropriate decision structure for a given situation
14.5	Select the correct nesting for decision structures
15.0 UTILIZE ITERATIVE STRUCTURES IN WRITING PROGRAMS	
15.1	Identify various types of iteration structure (e.g., while, for, for-each, recursion)
15.2	Identify how loops are controlled (variable conditions and exits)

15.3	Select the correct syntax for nested loops
15.4	Compute the values of variables involved with nested loops
16.0 UTILIZE PRIMITIVE DATA TYPES AND STRINGS	
16.1	Declare numeric, Boolean, character, and string variables
16.2	Choose the appropriate data type for a given situation
16.3	Identify the correct syntax for constants in a program
16.4	Identify the correct syntax for initializing and modifying variables
16.5	Identify the correct syntax and safe functions for operations on strings, including length, substring, and concatenation
16.6	Explain complications of storing and manipulating dates
17.0 UTILIZE BASIC DATA STRUCTURES IN PROGRAMS	
17.1	Demonstrate basic uses of arrays, including initialization, storage, and retrieval of values
17.2	Distinguish between arrays and hashmaps (associative arrays)
17.3	Identify techniques for declaring, initializing, and modifying user-defined data types
17.4	Search and sort data in an array
17.5	Create and use two-dimensional arrays
17.6	Describe the efficiency of different sorting algorithms (e.g., bubble, insertion, merge)
17.7	Describe the efficiency of linear vs. binary searches [e.g., o(n) or o(log n)]
18.0 IDENTIFY WAYS TO INPUT AND OUTPUT INFORMATION	
18.1	Identify correct input/output statements in a program
18.2	Choose the correct method of assigning input to variables, including data sanitization
18.3	Choose the correct method of outputting data, with formatting and escaping
18.4	Employ graphics methods to create images at specified locations
18.5	Choose correct GUI objects for input and output of data to the GUI interface (e.g., text boxes, labels, radio buttons, check boxes, dropdowns, list boxes)

19.0 USE EXTERNAL DATA SOURCES WITHIN A PROGRAM	
19.1	Input data from a sequential file and database
19.2	Output data to a sequential file and/or database
19.3	Append data to an existing file
19.4	Update files and/or databases
20.0 EMPLOY OBJECT-ORIENTED PROGRAMMING TECHNIQUES	
20.1	Make a distinction between an object and a class
20.2	Differentiate among inheritance, composition, and class relationships
20.3	Instantiate objects from existing classes
20.4	Read the state of an object by invoking accessor methods
20.5	Change the state of an object by invoking a modifier method
20.6	Determine the requirements for constructing new objects by reading the documentation
20.7	Create a user-defined class
20.8	Create a subclass of an existing class
21.0 EMPLOY RUN-TIME ERROR-HANDLING TECHNIQUES	
21.1	Identify run-time errors
21.2	Describe error-handling strategies
21.3	Handle unexpected return values
21.4	Handle (catch) run-time errors and take appropriate action
21.5	Throw standard exception classes
21.6	Develop and throw custom exception classes