IDAHO
Division of Career Technical Education

# 2025 Programming and Software Development

## Program Standards

### CONTENT STANDARD 1.0: PROFESSIONAL ORGANIZATIONS AND LEADERSHIP

#### Performance Standard 1.1: Student Leadership in Career Technical Student Organizations (CTSO) and Professional Associations

1.1.1 Explore the role of professional organizations and/or associations in the Programming and Software Development industry.
1.1.2 Define the value, role, and opportunities provided through career technical student organizations.
1.1.3 Engage in career exploration and leadership development

### CONTENT STANDARD 2.0: INDUSTRY PRACTICES

#### Performance Standard 2.1: Essential Skills

2.1.1 Compare programming paradigms including procedural and object-oriented programming.
2.1.2 Decompose complex problems into simpler, more manageable problems.
2.1.3 Plan structure and procedures before writing programs.
2.1.4 Write readable code following industry practices (e.g., white space, naming conventions, comments).
2.1.5 Write syntactically correct statements.
2.1.6 Navigate a computer file system.
2.1.7 Reference documentation (e.g., language, library, framework) to use implementation details.
2.1.8 Write software based on customer specifications.

#### Performance Standard 2.2: Project Development

2.2.1 Compare software development lifecycles (e.g., Agile, Waterfall).
2.2.2 Describe project scope and scope creep.
2.2.3 Initialize or clone a repository, using source control (e.g., git).
2.2.4 Commit and push code, using source control (e.g., git).
2.2.5 Pull code, using source control (e.g., git).

### CONTENT STANDARD 3.0: DATA

#### Performance Standard 3.1: Variables and Data Types

3.1.1 Identify the scope of a given variable.
3.1.2 Identify the value of a variable at a given point.
3.1.3 Declare and instantiate variables.
3.1.4 Reassign a variable.
3.1.5 Write code that uses primitive data types (e.g., integer, floating points, boolean, character).
3.1.6 Write code that uses reference types (e.g., string, object, array).
3.1.7 Write code that uses operators (e.g., +, -, *, /, %).
3.1.8 Cast data types.
3.1.9 Compare primitive types and derived/reference types.
3.1.10 Define *constants* and *enumerations*.

#### Performance Standard 3.2: Arrays

3.2.1 Declare an array and assign values to array elements.
3.2.2 Access data stored in array elements.

| | |
|---|---|
| 3.2.3 | Iterate through all elements in an array. |
| 3.2.4 | Access data stored in array elements. |
| 3.2.5 | Create multidimensional arrays. |
| 3.2.6 | Sort elements in an array. |
| **CONTENT STANDARD 4.0: CONTROL FLOW** | |
| Performance Standard 4.1: Branching and Logic | |
| 4.1.1 | Execute decisions in a program, using “if,” “else-if,” and “else” statements. |
| 4.1.2 | Compare values with conditional operators (i.e., >, <, >=,<=, ==, !=). |
| 4.1.3 | Create compound conditional statements with logical operators (e.g., ! [NOT], && [AND], \|\| [OR]). |
| 4.1.4 | Execute decisions in a program, using a nested IF statement. |
| 4.1.5 | Execute decisions in a program, using the switch statement. |
| Performance Standard 4.2: Loops | |
| 4.2.1 | Create loops, using the while statement. |
| 4.2.2 | Create loops, using the for statement. |
| 4.2.3 | Write code that uses nested loops. |
| 4.2.4 | Write code that uses accumulators (e.g., running total, collection). |
| Performance Standard 4.3: Functions | |
| 4.3.1 | Describe reasons for writing functions (e.g., to improve readability, reusability, maintainability). |
| 4.3.2 | Write functions with no parameters and no return value. |
| 4.3.3 | Write functions that require one or more parameters. |
| 4.3.4 | Write functions that return a value. |
| 4.3.5 | Call functions. |
| 4.3.6 | Pass parameters to functions. |
| 4.3.7 | Write code that uses return values from functions. |
| 4.3.8 | Import libraries. |
| 4.3.9 | Write a recursive function. |
| **CONTENT STANDARD 5.0: INPUT, DEBUGGING, AND EXCEPTIONS** | |
| Performance Standard 5.1 Input and Output | |
| 5.1.1 | Write a program that produces intended output. |
| 5.1.2 | Provide appropriate prompts for user input. |
| 5.1.3 | Take input from a user. |
| 5.1.4 | Take input from a file. |
| 5.1.5 | Validate input. |
| 5.1.6 | Write to a file. |
| Performance Standard 5.2: Debugging | |
| 5.2.1 | Debug programs by printing values to the console. |
| 5.2.2 | Inspect program state at runtime, using a debugger (e.g., breakpoints, stepping through code). |
| 5.2.3 | Inspect variable values during runtime, using a debugger. |
| 5.2.4 | Identify the contents of the call stack. |
| 5.2.5 | Fix syntax and logic errors. |
| 5.2.6 | Test applications, using varied input. |
| Performance Standard 5.3: Exception Handling | |
| 5.3.1 | Catch exceptions. |
| 5.3.2 | Write code that uses the finally block. |
| 5.3.3 | Throw exceptions. |
| **CONTENT STANDARD 6.0: OBJECT-ORIENTED PROGRAMMING** | |
| Performance Standard 6.1: Classes and Objects | |
| 6.1.1 | Define *abstraction*. |

| | |
|---|---|
| 6.1.2 | Describe object-oriented programming. |
| 6.1.3 | Create classes and instantiate objects from those classes. |
| 6.1.4 | Create properties. |
| 6.1.5 | Write constructors. |
| 6.1.6 | Describe public and private access (e.g., variables, methods). |
| 6.1.7 | Overload methods and constructors. |
| 6.1.8 | Reference the current object instance inside a class method (e.g., “this” in Java). |
| 6.1.9 | Demonstrate inheritance (“is a” relationships) by extending classes. |
| 6.1.10 | Demonstrate composition (“has a” relationships) by using a class object as a property in another class. |
| 6.1.11 | Demonstrate polymorphism by overriding parent class methods. |
| 6.1.12 | Implement interfaces. |
| Performance Standard 6.2: Events | |
| 6.2.1 | Define and apply event handling. |
| 6.2.2 | Handle control component events. |
| 6.2.3 | Handle mouse and keyboard events. |