STRANDS AND STANDARDS

COMPUTER PROGRAMMING 2

Course Description

This course introduces students to more advanced programming concepts. Students will learn to create more powerful programs within a specific programming language such as Java, Python, C++, or C#.

Intended Grade Level	9-12
Units of Credit	0.5
Core Code	35-02-00-00-032
Concurrent Enrollment Core Code	35-02-00-13-032
Prerequisite	Computer Programming 1 or Teacher Approval
Skill Certification Test Number	822 - C++ 827 - Python 828 - C# 824 - Java
Test Weight	0.5
License Area of Concentration	CTE and/or Secondary Education 6-12
Required Endorsement(s)	
Endorsement 1	Intro to Computer Science
Endorsement 2	Programming & Software Development
Endorsement 3	N/A

STRAND 1

Students will demonstrate static (array), dynamic (vector, Arraylist, etc.) list structures, and strings.

Standard 1

Demonstrate the ability to use static arrays/lists in programs.

- Declare and initialize arrays/lists of all applicable types
- Perform data input to and output from arrays/lists
- Perform operations on arrays/lists including sort arrays
- Iterate through the structure (i.e., for-each, enhanced for, or iterators)

Standard 2

Demonstrate the ability to use dynamic arrays/lists (i.e. vectors, Arraylist, or generic lists).

- Declare and initialize a dynamic array/list
- Add and remove items from the array/list
- Output data from arrays/lists
- Perform operations on arrays/lists
- Iterate through the structure (i.e. for-each, enhanced for, or iterators)
- Use a loop to iterate through the structure

Standard 3

Demonstrate the ability to use strings in programs.

- Compare string values
- Find the length of a string
- Copy part or all of string values into other strings
- Concatenate string values
- Locate substring positions
- Insert strings into other strings

Performance Skills

Students demonstrate mastery of static and dynamic arrays, lists, and strings in projects.

STRAND 2

Students will properly use sequential files.

Standard 1

Demonstrate the ability to use sequential files in programs.

- Create and initialize sequential files
- Store data to sequential files
- Retrieve data from sequential files
- Update sequential files

Performance Skills

Demonstrate sequential file access utilizing reading and writing operations.

STRAND 3

Create user defined functions using top-down design and functional decomposition.

Standard 1

Students will understand and properly apply scope.

- Understand that variables and functions have scope, which influences where they can be declared and accessed
- Declare and access local variables in a program
- Declare and access global variables in a program

Standard 2

Students will understand and implement function inputs and outputs.

- Understand the correlation between arguments (inputs) and parameters (variables)
- Understand that functions may or may not require arguments
- Understand that functions may or may not return values
- Define function(s):
  - with parameters
  - without parameters
  - with return values
  - without return values
  - default parameters

Standard 3

Students will understand and implement functional decomposition. (Breaking a program down into one or more functions.)

- Identify repetitive or redundant code in an application
- Understand the role abstraction plays in computer programming
- Demonstrate how to abstract multiple steps into a function
- Identify the characteristics of a well-defined function
  - Examples: shorter code, efficiency, reduced memory consumption, high reliability, readability, abstraction

Performance Skills

Create several user defined functions with and without inputs and/or return values.

STRAND 4

Students will properly demonstrate object-oriented programming techniques.

Standard 1

Demonstrate the ability to use built-in classes.

- Instantiate objects
- Use object data members (i.e., Java's arr. length)
- Use object member functions (methods)

Standard 2

Demonstrate the ability to create user-defined classes.

- Create and use data members (instance variables)
- Create a constructor to initialize the data members

- Create and use member functions (methods)

Performance Skills
Properly employ object-oriented programming techniques.

STRAND 5
Students will properly demonstrate code comprehension and debugging techniques.

Standard 1
Demonstrate the ability to comprehend code outcomes.
- Tracing - Cognitively following the passes of a loop, nested function calls, change in value of global and local scoped variables, etc.
- Debugging - Utilizing 3rd party tools (IDE's) to step through a program and troubleshoot
- Testing - Validating the outputs of a program and testing its robustness. (i.e., boundary conditions, invalid inputs, unexpected scenarios, incorrect results, etc.)

Performance Skills
Demonstrate code comprehension and debugging techniques by tracing, debugging, and testing programs.

STRAND 6
Students will apply appropriate programming skill as an effective member of a team demonstrating the ability to collaborate with others (www.p21.org).

Standard 1
Demonstrate the ability to apply knowledge to a programming project.
- Formalize specifications
- Choose proper input parameters
- Choose appropriate data structures and processing
- Design appropriate output
- Use appropriate test data
- Write good documentation

Standard 2
Demonstrate the ability to use teamwork and collaboration in a programming project.
- Divide a project among programmers
- Present work to a group
- Coordinate work with others in the group
- Complete assigned work according to predetermined deadlines
- Participate in a peer performance evaluation
- Demonstrate professionalism in team relationships, communication, timeliness, and attitude

Performance Skills
Apply appropriate programming skills as an effective member of a team.

STRAND 7

Students will demonstrate knowledge of current ethical issues dealing with computers and information in a global society using 21st Century Skills.

Standard 1

Demonstrate knowledge of the social and ethical consequences of computers.

- Explain the ethical reasons for creating reliable and robust software
- Explain the impact software can have on society (i.e., privacy, piracy, copyright laws, ease of use, ete.)
- Show how security concerns can be addressed in an application (i.e., biometrics, passwords, information hiding, etc.)
- Describe how computer-controlled automation affects a workplace and society
- Give examples of ways to protect information on computer systems (attacks, viruses, malware, etc.)

Performance Skills

Demonstrate knowledge of current ethical issues dealing with computers and information in society.

STRAND 8

Students will be aware of career opportunities in the Computer Programming/Software Engineering industry and of its history.

Standard 1

Investigate career opportunities, trends, and requirements related to computer programming/software engineering careers.

- Identify the members of a computer programming/software engineering team: team leader, analyst, senior developer, junior developer, and client/subject matter expert
- Describe work performed by each member of the computer programming/software engineering team
- Investigate trends and traits associated with computer programming/software engineering careers (creativity, technical, leadership, collaborative, problem solving, design, etc.)
- Discuss related career pathways

Performance Skills

Develop awareness of career opportunities in the computer programming/software engineering industry and of its history.

Workplace Skills

Workplace Skills taught:

- Communication
- Problem Solving
- Teamwork
- Critical Thinking
- Dependability
- Accountability
- Legal requirements / expectations

Skill Certificate Test Points by Strand

Test Name	Test #	Number of Test Points by Strand										Total Points	Total Questions
		1	2	3	4	5	6	7	8	9	10		
Computer Programming 2 C++	822	11	3	9	7	7	2	0	1			44	40
Computer Programming 2 JAVA	824	11	3	9	7	7	2	0	1			44	40
Computer Programming 2 PYTHON	827	12	2	9	7	7	2	0	1			42	40
Computer Programming 2 C#	828	12	2	10	7	6	2	0	1			44	40

Computer Programming 2 Vocabulary

Strand 1 - Students will demonstrate static (Array), dynamic(Vector, Array/List, etc.) list structures, and strings.	
Declaration	Stating the name and data type of a variable.
Initialization	Assignment of an initial value for a variable.
Iterate	Each cycle through a loop.
Dynamic Array/List	An Array/List that is able to change its size during program execution.
Static Array/List	Static arrays have their size or length determined when the array is created and/or allocated. For this reason, they may also be referred to as fixed-length arrays or fixed arrays.
Concatenate	Operation of joining two strings together.
Substring	Contiguous sequence of characters within a string.
Strand 2 - Students will properly use sequential files	
Sequential File	A sequential file **contains records organized by the order in which they were entered**. The order of the records is fixed. Records in sequential files can be read or written only sequentially.
Strand 3 - Create user defined functions using top-down design and functional decomposition.	
Scope	Determines the accessibility (visibility) of variables.
Local Variable	Only recognized inside the function in which it is declared.
Global Variable	Recognized from anywhere inside a program.
Arguments	The variables given to the function for execution.(Inputs)
Parameters	The names listed in the method/function's definition.(Variables)
Return	A value that is sent back to the user by a method/function.

Strand 4 - Students will properly demonstrate object-oreinted programming techniques.	
Method (There are both user defined methods and built in language specific methods)	A method is a programmed procedure that is defined as part of a class and included in any object of that class
Instance Variables	If the value of a variable varies from object to object, then such variables are called instance variables.
Data Members	A class variable or instance variable that holds data associated with a class and its objects
Instantiate Objects	The combined process of “make me a new object” and “get its settings initialized to the factory default settings” is called instantiation
Constructor	A constructor is **a special method of a class or structure in object-oriented programming that initializes a newly created object of that type**. Whenever an object is created, the constructor is called automatically.
Strand 5 - Students will properly demonstrate code comprehension and debugging techniques.	
IDE	An integrated development environment (IDE) is software for building applications that combines common developer tools into a single graphical user interface (GUI).
Strand 6 - Students will apply appropriate programming skills as an effective member of a team demonstrating the ability to collaborate with others(www.p21.org).	
Strand 7 - Students will demonstrate knowledge of current ethical issues dealing with computers and information in a global society using 21st Century skills.	
Strand 8 - Students will be aware of career opportunities in the Computer Programming/Software Engineering and of its history.	
Computer Programming/Software Engineering Team	Team Leader Analyst Senior Developer Junior Developer Client/Subject-Matter Expert

Skills Reference Sheet

Assignment, Display, and Input	
`a = expression`	Evaluates `expression` and then assigns a copy of the result to the variable **a**.
`DISPLAY(expression)`	Displays the value of `(expression)` in the console window.
`INPUT( )`	Accepts a value from the user and returns the input value.
Arithmetic Operators and Numeric Procedures	

`a + b`
`a - b`
`a * b`
`a / b`
	The arithmetic operators `+`, `–`, `*`, and `/` are used to perform arithmetic on `a` and `b`.
	
	For example, `17 / 5` evaluates to `3.4`.
	
	The order of operations used in mathematics applies when evaluating expressions.

`a MODULUS b`
`-or-`
`a MOD b`
	Evaluates to the remainder when `a` is divided by `b`.
	
	For example, `17 MOD 5` evaluates to `2`.
	
	`MODULUS (MOD)` has the same precedence as the `*` and `/` operators.

Relational and Boolean Operators	
`NOT condition`	Evaluates to `true` if `condition` is `false`; otherwise evaluates to `false`.
`condition1 AND condition2`	Evaluates to `true` if both `condition1` and `condition2` are `true`; otherwise evaluates to `false`.
`condition1 OR condition2`	Evaluates to `true` if `condition1` is `true` or if `condition2` is `true` or if both `condition1` and `condition2` are `true`; otherwise evaluates to `false`.

`FOR(condition)`
`{`
`  <block of statements>`
`}`
	The code in `<block of statements>` is executed a certain number of times.

`WHILE(condition)`
`{`
`  <block of statements>`
`}`
	The code in `<block of statements>` is repeated until the `(condition)` evaluates to `false`.

`IF(condition1)`
`{`
`  <first block of statements>`
`{`
`ELSE IF(condition2)`
`{`
`  <second block of statements>`
`}`
`ELSE`
`{`
`  <third block of statements>`
`}`
	If `(condition1)` evaluates to `true`, the code in `<first block of statements>` is executed; if `(condition1)` evaluates to `false`, then `(condition2)` is tested; if `(condition2)` evaluates to `true`, the code in `<second block of statements>` is executed; if both `(condition1)` and `(condition2)` evaluate to `false`, then the code in `<third block of statements>` is executed.

Procedures and Procedure Calls	

`PROCEDURE procName( )`
`{`
`  <block of statements>`
`}`
	Defines `procName` as a procedure that takes no arguments. The procedure contains `<block of statements>`.
	
	The procedure `procName` can be called using the following notation:
	
	`procName( )`