Introduction

There are three common sets of standards that high school computer science teachers in Arizona engage with:

- The Arizona Computer Science Standards developed by the Arizona Department of Education in 2018
- The Arizona CTE Software and App Design Standards developed by the Arizona Department of Education and updated in 2017
- The AP Computer Science Principles (AP CSP) Framework developed by the College Board and updated in 2020

This document illustrates how these standards overlap and are aligned, and illuminate areas where there may be holes that require supplemental instruction. This document was prepared by current Arizona educators who teach computer science courses that incorporate the CTE Software & App Design standards, with support from the Arizona Science Center and the Arizona Computer Science Teachers Association (CSTA-Arizona).

In this document, the **Arizona Computer Science Standards** are presented in order along with the CTE and AP CSP standards they align to. This may be useful for teachers who have already aligned a course to the Arizona Computer Science standards and want to see how this course is also mapped to the CTE or AP CSP framework.

An appendix is offered at the end of this document with guidance and next-steps for teachers who would like to continue aligning their courses between these sets of standards.

Standard Alignment

AZ Computer Science Standards		AZ CTE - Software & App Design Standards	AP CSP Framework
Computing Systems			
HS.CS.D.1	Explain how abstractions hide the underlying implementation details of computing systems embedded in everyday objects.	9.6 - Identify Internet of Things (IOT) and common communication interfaces (e.g., bluetooth, NFC, Wi-Fi, LTE)	AAP-3.B Explain how the use of procedural abstraction manages complexity in a program AAP-1.D For data abstraction: a. Develop data abstraction using lists to store multiple elements b. Explain how the use of data abstraction manages complexity in program code
HS.CS.HS.1	Describe levels of abstraction and interactions between application software, system software, and hardware layers.	5.1 Apply basic mathematics to hardware (e.g., bits, bytes, kilobytes, megabytes, gigabytes, and terabytes 5.2 Use binary to decimal, decimal to hexadecimal, hexadecimal to decimal, binary to hexadecimal, and binary to hexadecimal conversions to solve hardware and software problems	AAP-3.B Explain how the use of procedural abstraction manages complexity in a program
HS.CS.T.1	Develop guidelines that convey systematic troubleshooting strategies that others can use to identify and fix errors.	1.4 - Describe problem-solving and troubleshooting strategies applicable to software development	CRD-2.I For errors in an algorithm or program: a. Identify the error. b. Correct the error CRD-2.J Identify inputs and corresponding expected outputs or behaviors that can be used to check the correctness of an algorithm or program.

Networks and Internet			
HS.NI.C.1	Describe how sensitive data can be affected by malware and other attacks.	2.1 - Identify common computer threats (e.g., viruses, phishing, suspicious email, social engineering, spoofing, identity theft, spamming) 2.2 - Describe potential vulnerabilities in software (e.g., OWASP's Top 10) 2.3 - Identify procedures to maintain data integrity and security (e.g., lock the screen, delete unrecognized emails, use trustworthy thumb drives, use approved software) 2.6 - Explain the CIA (confidentiality, integrity, and availability) triad 3.3 - Identify issues and regulations affecting computers, other devices, the internet, and information privacy (e.g., HIPAA, COPPA, CISPA, FERPA, PCI, GDPR, data brokers)	IOC-2.A Describe the risks to privacy from collecting and storing personal data on a computer system. IOC-2.C Explain how unauthorized access to computing resources is gained.
HS.NI.C.2	Recommend security measures to address various scenarios based on factors such as efficiency, feasibility, and ethical impacts.	2.3 - Identify procedures to maintain data integrity and security (e.g., lock the screen, delete unrecognized emails, use trustworthy thumb drives, use approved software) 2.5 - Describe methods for sanitizing user input to prevent issues (e.g., buffer overflows, SQL injection) 2.7 - Explain how software defects relate to software security (e.g., buffer overflows, cross-site scripting) 3.1 Explore intellectual property rights including software licensing and software duplication [e.g., Digital Millennium Copyright Act (DMCA), software licensing, and software duplication]	IOC-2.A Describe the risks to privacy from collecting and storing personal data on a computer system. IOC-2.B Explain how computing resources can be protected and can be misused. IOC-2.C Explain how unauthorized access to computing resources is gained.

HS.NI.C.3	Compare various security measures, considering tradeoffs between the usability and security of a computing system.	2.2 - Describe potential vulnerabilities in software (e.g., OWASP's Top 10) 2.3 - Identify procedures to maintain data integrity and security (e.g., lock the screen, delete unrecognized emails, use trustworthy thumb drives, use approved software) 2.4 - Explain best practices to maintain integrity and security in software development (e.g., encryption, hashing, digital signatures) 2.5 - Describe methods for sanitizing user input to prevent issues (e.g., buffer overflows, SQL injection) 2.7 - Explain how software defects relate to software security (e.g., buffer overflows, cross-site scripting) 3.2 - Compare and contrast open source and proprietary systems in relation to legal and ethical issues (e.g., data pricing, use of public and private networks, social networking, industry-related data, data piracy)	IOC-2.A Describe the risks to privacy from collecting and storing personal data on a computer system. IOC-2.B Explain how computing resources can be protected and can be misused. IOC-2.C Explain how unauthorized access to computing resources is gained.
HS.NI.NCO.1	Evaluate the scalability and reliability of networks, by describing the relationship between routers, switches, servers, topology, and addressing.	9.1 - Explain cloud-based computing and content delivery networks 9.2 - Identify the components and functions of the internet (e.g., HTTP, HTTPS, FTP, IP addresses, IMAP) 9.3 - Identify services run by web servers [e.g., scripting languages (client and server-side scripting), databases, media]	CSN-1.A Explain how computing devices work together in a network. CSN-1.B Explain how the Internet works CSN-1.C Explain how data are sent through the Internet via packets. CSN-1.D Describe the differences between the Internet and the World Wide Web.

	9.4 - Identify performance issues (e.g., bandwidth, internet connection types, pages loading slowly, resolution and size graphics) 9.5 - Differentiate among shared hosting, dedicated server, and virtual private server (VPS) 9.6 - Identify Internet of Things (IOT) and common communication interfaces (e.g., bluetooth, NFC, Wi-Fi, LTE)	CSN-1.E For fault-tolerant systems, like the Internet: a. Describe the benefits of fault tolerance. b. Explain how a given system is fault-tolerant. c. Identify vulnerabilities to failure in a system. CSN-2.A For sequential, parallel, and distributed computing: a. Compare problem solutions. b. Determine the efficiency of solutions CSN-2.B Describe benefits and challenges of parallel and distributed computing.
Data and Analysis		
HS.DA.CVT.1 Create interactive data visualizations using software tools to help others better understand real-world phenomena.	1.2 - Explain the process of decomposing a large programming problem into smaller, more manageable procedures 1.3 - Explain "visualizing" as a problem-solving technique prior to writing code 17.1 Input/output data from a sequential file or database	CRD-2.B Explain how a program or code segment functions CRD-2.C Identify input(s) to a program. CRD-2.D Identify output(s) produced by a program. CRD-2.E Develop a program using a development process CRD-2.F Design a program and its user interface. CRD-2.G Describe the purpose of a code segment or program by writing documentation. CRD-2.H Acknowledge code segments used from other sources.

HS.DA.S.1	Translate between different bit representations of real-world phenomena, such as characters, numbers, and images.	4.1 - Declare numeric, Boolean, character, string variables, and float and double 5.1 - Apply basic mathematics to hardware (e.g., bits, bytes, kilobytes, megabytes, gigabytes, terabytes) 5.2 - Use binary to decimal, decimal to hexadecimal, hexadecimal to decimal, binary to hexadecimal, and binary to hexadecimal conversions to solve hardware and software problems	DAT-1.A Explain how data can be represented using bits DAT-1.B Explain the consequences of using bits to represent data. DAT-1.C For binary numbers: a. Calculate the binary (base 2) equivalent of a positive integer (base 10) and vice versa. b. Compare and order binary numbers AAP-4.A For determining the efficiency of an algorithm: a. Explain the difference between algorithms that run in reasonable time and those that do not. b. Identify situations where a heuristic solution may be more appropriate.
HS.DA.S.2	Evaluate the tradeoffs in how and where data is stored.	3.3 - Identify issues and regulations affecting computers, other devices, the internet, and information privacy (e.g., HIPAA, COPPA, CISPA, FERPA, PCI, GDPR, data brokers) 4.2 - Choose the appropriate data type for a given situation 4.5 - Explain complications of storing and manipulating data, i.e., the Big-O notation for analyzing storage and efficiency concerns 9.4 - Identify performance issues (e.g., bandwidth, internet connection types, pages loading slowly, resolution and size graphics)	DAT-1.D Compare data compression algorithms to determine which is best in a particular context. DAT-2.A Describe what information can be extracted from data. DAT-2.C Identify the challenges associated with processing data. AAP-4.A For determining the efficiency of an algorithm: a. Explain the difference between algorithms that run in reasonable time and those that do not. b. Identify situations where a heuristic solution may be more appropriate.

HS.DA.IM.1	Analyze computational models to better understand real-world phenomena.	4.5 - Explain complications of storing and manipulating data, i.e., the Big-O notation for analyzing storage and efficiency concerns 8.6 - Describe the efficiency of different sorting algorithms (e.g., bubble, insertion, merge) 8.7 - Describe the efficiency of linear vs. binary searches [e.g., o(n) or o(log n)]	AAP-3.F For simulations: a. Explain how computers can be used to represent real-world phenomena or outcomes b. Compare simulations with real-world contexts. AAP-4.B Explain the existence of undecidable problems in computer science DAT-2.D Extract information from data using a program DAT-2.E Explain how programs can be used to gain insight and knowledge from data
Algorithms and Programming			
HS.AP.A.1	Create prototypes that use algorithms for practical intent, personal expression, or to address a societal issue	4.3 - Identify the correct syntax and usage for constants and variables in a program 5.3 - Identify and correctly use arithmetic operations, applying the order of operations (precedence) with respect to programming 11.3 - Apply pseudocode or graphical representations to plan the structure of a program or module (e.g., flowcharting, white boarding, UML) 11.4 - Create and implement basic algorithms	CRD-2.F Design a program and its user interface
HS.AP.V.1	Use lists to simplify solutions, generalizing computational problems instead of repeatedly using simple variables.	4.3 - Identify the correct syntax and usage for constants and variables in a program 8.1 - Demonstrate basic uses of arrays, including initialization, storage, and retrieval of values 8.2 - Distinguish between arrays and hash maps (associative arrays) 8.3 - Identify techniques for declaring, initializing, and modifying user-defined data types 8.4 - Search and sort data in an array 8.5 - Create and use two-dimensional arrays	AAP-2.N For list operations: a. Write expressions that use list indexing and list procedures. b. Evaluate expressions that use list indexing and list procedures. AAP-2.O For algorithms involving elements of a list: a. Write iteration statements to traverse a list. b. Determine the result of an algorithm that includes list traversals. AAP-2.P For binary search algorithms: a. Determine the number of iterations required to find a value in a data set. b. Explain the requirements necessary to complete a

		7.1 - Identify various types of iteration structure (e.g., while, for, for-each, recursion) 7.2 - Identify how loops are controlled (variable conditions and exits) 7.3 - Use the correct syntax for nested loops 7.4 - Compute the values of variables involved with nested loops	binary search
HS.AP.C.1	Justify the selection of specific control structures and explain the benefits and drawbacks of choices made, when tradeoffs involve readability and program performance.	4.3 - Identify the correct syntax and usage for constants and variables in a program 6.1 - Use the correct syntax for decision statements (e.g., if/else, if, switch case) 6.2 - Compare values using relational operators (e.g., =, >, <, >=, <=, not equal) 6.3 - Evaluate Boolean expressions (e.g., AND, OR, NOT, NOR, XOR) 6.4 - Use the correct nesting for decision structures 7.1 - Identify various types of iteration structure (e.g., while, for, for-each, recursion) 7.2 - Identify how loops are controlled (variable conditions and exits) 7.3 - Use the correct syntax for nested loops 7.4 - Compute the values of variables involved with nested loops	AAP-2.E For relationships between two variables, expressions, or values: a. Write expressions using relational operators. b. Evaluate expressions that use relational operators. AAP-2.F For relationships between Boolean values: a. Write expressions using logical operators. b. Evaluate expressions that use logic operators AAP-2.H For selection: a. Write conditional statements. b. Determine the result of conditional statements AAP-2.L Compare multiple algorithms to determine if they yield the same side effect or result. AAP-2.M For algorithms: a. Create algorithms. b. Combine and modify existing algorithms CRD-2.G Describe the purpose of a code segment or program by writing documentation. CRD-2.I For errors in an algorithm or program: a. Identify the error. b. Correct the error.

HS.AP.C.2	Use events that initiate instructions to design and iteratively develop computational artifacts	4.3 - Identify the correct syntax and usage for constants and variables in a program 12.1 - Use a program editor to enter and modify code 12.2 - Identify correct input/output statements 12.3 - Choose the correct method of assigning input to variables including data sanitization 12.4 - Choose the correct method of outputting data with formatting and escaping 12.7 - Apply industry standards in documentation (e.g., self-documenting code; function-level, program-level, and user-level documentation) 13.1 - Identify errors in program modules 13.4 - Categorize, identify, and correct errors in code, including syntax, semantic, logic, and runtime 13.5 - Perform different methods of debugging (e.g., hand-trace code and real time debugging tools) 16.5 - 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)	AAP-3.A For procedure calls: a. Write statements to call procedures. b. Determine the result or effect of a procedure call
HS.AP.M.1	Decompose problems into smaller components using constructs such as procedures, modules, and/or objects.	1.2 - Explain the process of decomposing a large programming problem into smaller, more manageable procedures 11.4 - Create and implement basic algorithms 12.9 - Demonstrate refactoring techniques to reduce repetitious code and improve maintainability 12.10 - Demonstrate the use of parameters to pass data into program modules 12.11 - Demonstrate the use of return values from modules	AAP-2.A Express an algorithm that uses sequencing without using a programming language. AAP-2.B Represent a step-by-step algorithmic process using sequential code statements

HS.AP.M.2	Use procedures within a program, combinations of data and procedures, or independent but interrelated programs to design and iteratively develop computational artifacts.	4.3 - Identify the correct syntax and usage for constants and variables in a program 12.1 - Use a program editor to enter and modify code 12.2 - Identify correct input/output statements 12.10 - Demonstrate the use of parameters to pass data into program modules 12.11 - Demonstrate the use of return values from modules 13.1 - Identify errors in program modules 13.2 - Identify boundary cases and generate appropriate test data 14.1 - Use standard library functions 14.2 - Find and use third party libraries (e.g., web-based and package managers) 14.3 - Explain and interact with an Application Program Interface (API)	AAP-2.C Evaluate expressions that use arithmetic operators AAP-2.D Evaluate expressions that manipulate strings. AAP-2.E For relationships between two variables, expressions, or values: a. Write expressions using relational operators. b. Evaluate expressions that use relational operators For relationships between Boolean values: a. Write expressions using logical operators. b. Evaluate expressions that use logic operators AAP-2.H For selection: a. Write conditional statements. b. Determine the result of conditional statements AAP-2.I For nested selection: a. Write nested conditional b. Determine the result of nested conditional statements. AAP-2.K For iteration: a. Write iteration statements. b. Determine the result or side effect of iteration statements.
HS.AP.PD.1	Evaluate and refine computational artifacts to make them more usable and accessible.	11.2 - Develop program requirements/specifications and a testing plan (e.g., user stories, automated testing, test procedures) 12.1 - Use a program editor to enter and modify code 12.3 - Choose the correct method of assigning input to variables, including data sanitization 12.9 - Demonstrate refactoring techniques to reduce repetitious code and improve maintainability 13.1 - Identify errors in program modules 13.2 Identify boundary cases and generate appropriate test data	AAP-2.L Compare multiple algorithms to determine if they yield the same side effect or result. AAP-2.M For algorithms: a. Create algorithms. b. Combine and modify existing algorithms. AAP-2.N For list operations: a. Write expressions that use list indexing and list procedures. b. Evaluate expressions that use list indexing and list procedures. AAP-2.O For algorithms involving elements of a list: a. Write iteration statements to traverse a list. b. Determine the result of an algorithm that includes list traversals.

		13.3 - Perform integration testing including tests within a program to protect execution from bad input or other run-time errors 13.4 - Categorize, identify, and correct errors in code, including syntax, semantic, logic, and runtime 13.5 - Perform different methods of debugging (e.g., hand-trace code or real time debugging tools)	AAP-2.P For binary search algorithms: a. Determine the number of iterations required to find a value in a data set. b. Explain the requirements necessary to complete a binary search. CRD-2.I For errors in an algorithm or program: a. Identify the error. b. Correct the error. CRD-2.J Identify inputs and corresponding expected outputs or behaviors that can be used to check the correctness of an algorithm or program.
HS.AP.PD.2	Use team roles and collaborative tools to design and iteratively develop computational artifacts.	1.1 - Establish objectives and outcomes for a task 1.2 - Explain the process of decomposing a large programming problem into smaller, more manageable procedures 1.4 - Describe problem-solving and troubleshooting strategies applicable to software development 10.2 - Use client collaboration sources/platforms (e.g., GitHub, Google Drive, Dropbox, jsfiddle, browser developer tools)	CRD-1.A Explain how computing innovations are improved through collaboration. CRD-1.B Explain how computing innovations are developed by groups of people. CRD-1.C Demonstrate effective interpersonal skills during collaboration.
HS.AP.PD.3	Document design decisions using text, graphics, presentations, and/or demonstrations in the development of complex programs.	1.3 - Explain "visualizing" as a problem-solving technique prior to writing code 11.1 - Implement the steps in the System Development Life Cycle (SDLC) (e.g., planning, analysis, design, development, testing, implementation, maintenance) 11.2 - Develop program requirements/specifications and a testing plan (e.g., user stories, automated testing, test procedures) 11.3 - Apply pseudocode or graphical representations to plan the structure of a program or module (e.g., flowcharting, white boarding, UML) 12.7 - Apply industry standards in documentation (e.g.,	CRD-2.E Develop a program using a development process. CRD-2.F Design a program and its user interface. CRD-2.G Describe the purpose of a code segment or program by writing documentation.

	self-documenting code; function-level, program-level, user-level documentation) 12.8 - Name identifiers and formatting code by applying recognized conventions 12.9 - Demonstrate refactoring techniques to reduce repetitious code and improve maintainability	
Impacts of Computing		
HS.IC.C.1 Evaluate the ways access to computing impacts personal, ethical, social, economic, and cultural practices.		IOC-1.A Explain how an effect of a computing innovation can be both beneficial and harmful. IOC-1.B Explain how a computing innovation can have an impact beyond its intended purpose. IOC-1.C Describe issues that contribute to the digital divide. IOC-1.F Explain how the use of computing can raise legal and ethical concerns.
HS.IC.C.2 Test and refine computational artifacts to reduce bias and equity deficits.		IOC-1.D Explain how bias exists in computing innovations. IOC-1.E Explain how people participate in problem solving processes at scale.
HS.IC.C.3 Demonstrate ways a given algorithm applies to problems across disciplines.		IOC-1.E Explain how people participate in problem solving processes at scale.
HS.IC.SI.1 Analyze the impact of collaborative tools and methods that increase social connectivity.	3.2 - Compare and contrast open source and proprietary systems in relation to legal and ethical issues (e.g., data pricing, use of public and private networks, social networking, industry-related data, data piracy) 10.2 - Use client collaboration sources/platforms (e.g., GitHub, Google Drive, Dropbox, jsfiddle, browser developer tools) 10.3 - Analyze remote computing tools and services and their application	IOC-1.E Explain how people participate in problem solving processes at scale.

HS.IC.SLE.1	Explain the beneficial and harmful effects that intellectual property laws can have on innovation.	3.1 - Explore intellectual property rights including software licensing and software duplication [e.g., Digital Millennium Copyright Act (DMCA), software licensing, software duplication] 3.3 - Identify issues and regulations affecting computers, other devices, the internet, and information privacy (e.g., HIPAA, COPPA, CISPA, FERPA, PCI, GDPR, data brokers)	IOC-1.F Explain how the use of computing can raise legal and ethical concerns IOC-2.A Describe the risks to privacy from collecting and storing personal data on a computer system. IOC-2.B Explain how computing resources can be protected and can be misused.
HS.IC.SLE.2	Explain the privacy concerns related to the collection and generation of data through automated processes that may not be evident to users.	3.2 - Compare and contrast open source and proprietary systems in relation to legal and ethical issues (e.g., data pricing, use of public and private networks, social networking, industry-related data, data piracy) 3.3 - Identify issues and regulations affecting computers, other devices, the internet, and information privacy (e.g., HIPAA, COPPA, CISPA, FERPA, PCI, GDPR, data brokers)	IOC-1.F Explain how the use of computing can raise legal and ethical concerns IOC-2.A Describe the risks to privacy from collecting and storing personal data on a computer system. IOC-2.B Explain how computing resources can be protected and can be misused.
HS.IC.SLE.3	Evaluate the social and economic implications of privacy in the context of safety, law, or ethics.	3.2 - Compare and contrast open source and proprietary systems in relation to legal and ethical issues (e.g., data pricing, use of public and private networks, social networking, industry-related data, data piracy) 3.3 - Identify issues and regulations affecting computers, other devices, the internet, and information privacy (e.g., HIPAA, COPPA, CISPA, FERPA, PCI, GDPR, data brokers) 10.3 - Analyze remote computing tools and services and their application	IOC-1.F Explain how the use of computing can raise legal and ethical concerns IOC-2.A Describe the risks to privacy from collecting and storing personal data on a computer system. IOC-2.B Explain how computing resources can be protected and can be misused.

Appendix: Aligning Between Courses

I am already teaching AP Computer Science Principles - how does this align with the Arizona Computer Science Standards?

Every entry for the Arizona Computer Science standards has a matching entry in the AP Computer Science Principles framework. Therefore, if you are teaching an approved AP Computer Science Principles curriculum, you are guaranteed to also address the Arizona Computer Science standards

I am already teaching a CTE course - how does this align with the Arizona Computer Science Standards?

Most CTE standards align with the Arizona Computer Science standards, but there are a few Arizona Computer Science standards on the Impact of Computing that do not appear in the CTE standards. You can watch this video on the Impacts of Computing standards to learn about additional resources or activities that can be used to supplement your class.

I am already teaching a course aligned with the Arizona Computer Science Standards - how does this align with the CTE standards?

The Software and App Design CTE standards are intended to cover a 2-year sequence of courses. The chart on the next page lists all of the CTE standards that are not addressed in the Arizona Computer Science standards. Many of these standards can be addressed in a separate class, taught before or after your current course. The following courses taught throughout Arizona address many of the additional CTE standards and can be used as an additional 2nd year course

- AP Computer Science A
- Microsoft TEALS Introduction to Computer Science
- Code.org Computer Science Discoveries
- Additional courses are listed in the Arizona CS Implementation Guide

CTE - Software & App Design Standards NOT included in Arizona Computer Science Standards

4.4 - Identify the correct syntax and safe functions for operations on strings, including length, substring, and concatenation
4.6 - Research industry relevant programming languages, i.e., Java, JavaScript, and Python
5.4 - Interpret and construct mathematical formulas
5.5 - Identify correct and problematic uses of integers, floating-point numbers, and fixed-point numbers in arithmetic
10.1 - Identify key components and functions of internet and web specialty browsers
12.5 - Differentiate between interpreted and compiled code (e.g., steps necessary to run executable code)
15.1 - Identify the purpose of version control systems (e.g., Git, Mercurial)
15.2 - Create a new repository
15.3 - Add, push, and pull source code from repository
15.4 - Explain branching and its uses
15.5 - Restore previous versions of code from the repository
16.1 - Apply W3C standards and style conventions
16.2 - Construct web pages and applications that are compliant with ADA and sections 504 and 508 standards
16.3 - Explain the concept of responsive design and applications
16.4 - Employ graphics methods to create images at specified locations
17.2 - Demonstrate creating, reading, updating, and dropping a database
17.3 - Demonstrate the proper use of SQL database applications that work with different languages, e.g., MongoDB, Microsoft Access, Oracle Databases, Code.org's AppLab)
18.1 - Make a distinction between an object and a class
18.2 - Differentiate among inheritance, composition, and class relationships
18.3 - Instantiate objects from existing classes
18.4 - Read the state of an object by invoking accessor methods
18.5 - Change the state of an object by invoking a modifier method
18.6 - Determine the requirements for constructing new objects by reading the documentation
18.7 - Create a user-defined class
18.8 - Create a subclass of an existing class
18.9 - Identify the use of an abstract class as opposed to an interface
18.1 - Explain the object-oriented concepts of polymorphism, inheritance, and encapsulation
19.1 - Identify run time errors
19.2 - Describe error handling strategies
19.3 - Handle unexpected return values
19.4 - Handle (catch) run time errors and take appropriate action
19.5 - Throw standard exception classes
19.6 - Develop and throw custom exception classes