Computer Science Principles 2020-2021
PLTW Course Level Framework

PLTW Framework - Overview

PLTW Frameworks are representations of the knowledge, skills, and understandings that empower students to thrive in an evolving world. The PLTW Frameworks define the scope of learning and instruction within the PLTW curricula. The framework structure is organized by four levels of understanding that build upon each other: Knowledge and Skills, Objectives, Domains, and Competencies.

The most fundamental level of learning is defined by course Knowledge and Skills statements. Each Knowledge and Skills statement reflects specifically what students will know and be able to do after they've had the opportunity to learn the course content. Students apply Knowledge and Skills to achieve learning Objectives, which are skills that directly relate to the workplace or applied academic settings. Objectives are organized by higher-level Domains.

Enduring understandings leave a lasting impression on students. Students build and earn these understandings over time by exploring and applying course content throughout the year.

Domains are areas of in-demand expertise that an employer in a specific field may seek; they are key understandings and long-term takeaways that go beyond factual knowledge into broader, conceptual comprehension.

At the highest level, Competencies are general characterizations of the transportable skills that benefit students in various professional and academic pursuits. As a whole, the PLTW Frameworks illustrate the deep and relevant learning opportunities students experience from PLTW courses and demonstrate how the courses prepare students for life, not just the next grade level.

To thrive in an evolving world, students need skills that will benefit them regardless of the career path they choose. PLTW Frameworks are organized to showcase alignment to in-demand, transportable skills. This alignment ensures that students learn skills that are increasingly important in the rapidly advancing, innovative workplace.

Competencies (C), Domains (D), Enduring Understandings (EU), Objectives (O), Knowledge and Skills (KS)

C1 Creativity and Problem-Solving

D1 Creativity

Computing is a creative activity. Creativity and computing are prominent forces in innovation; the innovations enabled by computing have had and will continue to have far-reaching impact.

O1.1 Apply a creative development process.

KS 1.1A Translate ideas into tangible form by creating computational artifacts and employing an iterative and exploratory process.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	☑	☑	□	☑	☑	□	☑	□	☑	□

KS 1.1B Discuss how creativity, collaboration, and curiosity can lead to innovation.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	☑	☑	□	☑	□	□	□	☑	□	☑	□

D2 Problem-Solving Mindset

There are professional characteristics and habits of action that help people create value for society though innovation and problem-solving.

O2.1 Acknowledge moments where persistence and the positive aspect of failure played an important role in gaining understanding about a problem.

KS 2.1A Describe difficulties and/or opportunities you encountered and how they were resolved or incorporated.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	☑	□	☑	□	☑	☑	☑	☑	□	□

O2.2 Engage stakeholders in a problem and use their perspectives to shape the course of your development.

KS 2.2A Identify programmer and user concerns that affect the solution to problems.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	□	□	□	□	□	□	☑	□	□	□	□

KS 2.2B Consult and communicate with program users in program development to solve problems.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	□	□	□	□	□	□	□	□

D3 Collaboration

Diverse perspectives, good interpersonal relationships, and effective collaboration strategies generate the most robust and innovative solutions.

O3.1 Collaborate when processing information to gain insight and knowledge.

KS 3.1A Understand that collaboration is an important part of solving data-driven problems.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	□	□	□	□	☑	☑	□	☑	□	□

KS 3.1B Understand that collaboration facilitates solving computational problems by applying multiple perspectives, experiences, and skill sets.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	☑	□	□	☑	□	☑	□	☑	☑	□

O3.2 Apply project management strategies effectively as part of a team.

KS 3.2A Prioritize short-term and long-term objectives using an Agile methodology when working on a project.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	☑	☑	□	□	□	□	□	□	☑	□	□

C2 Professional Practices and Communication

D4 Career Awareness

Today computing impacts almost all careers. There are career specializations within computer science, such as software development, security, network, and systems administration.

O4.1 Describe career paths within the computing specialties.

KS 4.1A Describe a variety of careers within the computing specialties.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	□	☑	□	☑	□	□	□	□	□	□	□

KS 4.1B Recognize the education and credentialing requirements for careers within computing specialties.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	□	□	□	□	□	□	□	□

KS 4.1C Demonstrate the initiative and independent learning required to stay current with evolving technology and career needs

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	□	□	□	□	□	□	☑	□

D5 Professionalism and Ethics

Computing professionals must make decisions regularly regarding their professional and social conduct when collaborating with developers and engaging with users to get feedback.

O5.1 Abide by professional standards when creating value for people and society.

KS 5.1A Abide by professional standards when creating value for people and society. Create and maintain a secure professional identity for accessing IDEs and accessing computer science communities.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	□	□	□	□	□	□	□	□

KS 5.1B Abide by professional standards when creating value for people and society. Provide rationales for all ethical decisions.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	□	□	□	□	□	□	☑	□

KS 5.1C Abide by professional standards when creating value for people and society. Engage others with respect and forethought.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	☑	□	□	□	□	☑	☑	☑	□	□

D6 Communication

Computing professionals must be able to explain and justify the design and appropriateness of their computational choices, and analyze and describe both computational artifacts and the results or behaviors of such artifacts.

O6.1 Communicate ideas, processes, and products to optimize audience perception and understanding.

KS 6.A Communicate ideas, processes, and products to optimize audience perception and understanding. Communicate which portions of a program you developed independently and which were created collaboratively.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	☑	☑	□	□	□	□	□	☑	□	□	□

C3 Big Ideas in Computer Science

D7 Creative Development

Incorporating multiple perspectives through collaboration improves computing innovations as they are developed. (EU)

Developers create and innovate using an iterative design process that is user-focused, that incorporates implementation/feedback cycles, and that leaves ample room for experimentation and risk-taking. (EU)

CRD-1.A Explain how computing innovations are improved through collaboration.

CRD-1.A.1 A computing innovation includes a program as an integral part of its function.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☐	☑	☑	☐	☐	☐	☐	☐	☑

CRD-1.A.2 A computing innovation can be physical (i.e., self-driving car), non-physical computing software (i.e., picture editing software), or non-physical computing concepts (i.e., eCommerce).

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☐	☑	☑	☐	☐	☐	☐	☐	☑

CRD-1.A.3 Effective collaboration produces a computing innovation that reflects the diversity of talents and perspectives of those who designed it.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☐	☑	☑	☐	☐	☑	☐	☐	☑

CRD-1.A.4 Collaboration that includes diverse perspectives helps to avoid bias in the development of computing innovations.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☐	☑	☑	☐	☐	☑	☐	☐	☑

CRD-1.A.5 Consultation and communication with users is an important aspect of the development of computing innovations.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☐	☑	☑	☐	☐	☑	☐	☐	☑

CRD-1.A.6 Research gathered from users can be used to understand the purpose of a program from diverse perspectives in order to develop a program that can be used to fully incorporate these perspectives.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☐	☑	☑	☐	☐	☐	☐	☐	☑

CRD-1.B Explain how computing innovations are developed by groups of people.

CRD-1.B.1 Online tools support collaboration by allowing programmers to virtually share and provide feedback on ideas and documents.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☐	☑	☑	☐	☐	☐	☐	☐	☐

CRD-1.B.2 Common models such as pair programming exist to facilitate collaboration.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☐	☑	☑	☐	☐	☑	☐	☐	☐

CRD-1.C Demonstrate effective interpersonal skills during collaboration.

CRD-1.C.1 Effective collaborative teams practice interpersonal skills including but not limited to:

- communication skills
- consensus building
- conflict resolution
- negotiation

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	☑	☑	☑	☑	☑	□	□	□

CRD-2.A Describe the purpose of a computing innovation.

CRD-2.A.1 The purpose of computing innovations is to solve problems or pursue interests through creative expression.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	☑	□	☑	☑	□	□	☑	□	□	□

CRD-2.A.2 An understanding of the purpose of a computing innovation provides developers with an improved ability to develop the computing innovation.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	☑	☑	□	□	☑	□	□	□

CRD-2.B Explain how a program or code segment functions.

CRD-2.B.1 A *program* is a collection of program statements that performs a specific task when run by a computer. A program is often referred to as *software*.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	□	□	□	□	□	□	□	□	□	□	□

CRD-2.B.2 A *code segment* refers to a collection of program statements that are part of a program.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	☑	□	□	□	□	☑	□	□	□	□

CRD-2.B.3 A program needs to work for a variety of inputs and situations.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	□	□	☑	□	□	☑	☑	☑	□	□	□

CRD-2.B.4 The *behavior* of a program is how a program functions during execution and is often described by how a user interacts with it.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	☑	☑	□	□	□	□	□	□	□	□

CRD-2.B.5 A program can be described broadly by what it does, or in more detail by both what the program does and how the program statements accomplish this function.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	☑	☑	□	□	□	□	☑	□	□	□

CRD-2.C Identify input(s) to a program.

CRD-2.C.1 *Program input* is data sent to a computer for processing by a program. Input can come in a variety of forms, such as tactile, audio, visual, or text.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	☑	□	☑	□	☑	☑	☑	□	□	□

CRD-2.C.2 An *event* is associated with an action and supplies input data to a program.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	☑	☑	☑	□	☑	☑	☑	□	□	□

CRD-2.C.3 Events can be generated when a key is pressed, a mouse is clicked, a program is started, or by any other defined action that affects the flow of execution.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	☑	☑	□	☑	□	☑	□	□	□	□	□

CRD-2.C.4 Inputs usually affect the output produced by a program.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	☑	☑	□	☑	□	☑	□	□	□	□	□

CRD-2.C.5 In event-driven programming, program statements are executed when triggered rather than through the sequential flow of control.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	☑	☑	☑	□	☑	☑	☑	□	□	□

CRD-2.C.6 Input can come from a user or other programs.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	☑	☑	□	☑	□	☑	□	☑	□	□	□

CRD-2.D Identify output(s) produced by a program.

CRD-2.D.1 *Program output* is any data that are sent from a program to a device. Program output can come in a variety of forms, such as tactile, audible, visual, or text.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	☑	□	☑	□	☑	☑	☑	□	□	□

CRD-2.D.2 Program output is usually based on a program’s input or prior state (e.g., internal values).

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	☑	☑	☑	□	☑	☑	☑	□	□	□

CRD-2.E Develop a program using a development process.

CRD-2.E.1 A development process could be ordered and intentional, or exploratory in nature.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	□	☑	□	☑	☑	☑	□	□	□	□	□

CRD-2.E.2 There are multiple development processes. The following phases are commonly used when developing a program:

- investigating and reflecting
- designing
- prototyping
- testing

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	☑	□	☑	☑	☑	□	☑	□	□	□

CRD-2.E.3 A development process that is iterative requires refinement and revision based on feedback, testing, or reflection throughout the process. This may require revisiting earlier phases of the process.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	☑	□	☑	☑	□	☑	☑	□	□	□

CRD-2.E.4 A development process that is incremental is one that breaks the problem into smaller pieces and makes sure each piece works before adding it to the whole.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	☑	□	☑	☑	☑	☑	☑	□	□	□

CRD-2.F Design a program and its user interface.

CRD-2.F.1 The design of a program incorporates investigation to determine the requirements.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	☑	☑	☑	☑	☑	□	☑	□	□	□

CRD-2.F.2 Investigation in a development process is useful in understanding and identifying the constraints, concerns, and interests of the people who will use the program.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	☑	□	☑	☑	□	☑	□	□	□

CRD-2.F.3 Some ways investigation can be performed are as follows:

- collecting data through surveys
- user testing
- interviews
- direct observations

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	☑	☑	☑	☑	□	□	□	□	□

CRD-2.F.4 Program requirements describe how a program functions and may include a description of user interactions that a program must provide.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	☑	☑	☑	☑	☑	□	□	□	□	□

CRD-2.F.5 A program's specification defines the requirements for the program.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	☑	☑	□	☑	☑	□	☑	□	□	□

CRD-2.F.6 In a development process, the design phase outlines how to accomplish a given program specification.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	☑	☑	□	☑	☑	□	☑	□	□	□

CRD-2.F.7 The design phase of a program includes:

- brainstorming
- planning and storyboarding
- organizing the program into modules and functional components
- creation of diagrams that represent the layouts of the user interface
- development of a testing strategy for the program

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	☑	☑	☑	☑	☑	□	☑	□	□	□

CRD-2.G Describe the purpose of a code segment or program by writing documentation.

CRD-2.G.1 *Program documentation* is a written description of the function of a code segment, event, procedure, or program and how it was developed.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	☑	☐	☐	☑	☐	☑	☑	☐	☐	☐

CRD-2.G.2 *Comments* are a form of program documentation written into the program to be read by people and do not affect how a program runs.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	☑	☐	☐	☑	☐	☑	☑	☐	☐	☐

CRD-2.G.3 Programmers should document a program throughout its development.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	☑	☐	☐	☑	☐	☐	☑	☐	☐	☐

CRD-2.G.4 Program documentation helps in developing and maintaining correct programs when working individually or in collaborative programming environments.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	☑	☐	☐	☐	☐	☐	☑	☐	☐	☐

CRD-2.G.5 Not all programming environments support comments, so other methods of documentation may be required.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐

CRD-2.H Acknowledge code segments used from other sources.

CRD-2.H.1 It is important to acknowledge any code segments that were developed collaboratively or by another source.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☐	☐	☐	☑	☐	☑	☑	☐	☐

CRD-2.H.2 Acknowledgement of code segment(s) written by someone else and used in a program can be in the program documentation. The acknowledgement should include the origin or original author's name.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☐	☐	☐	☑	☐	☑	☑	☐	☐

CRD-2.I For errors in a program: (a) Identify the error. (b) Correct the error.

CRD-2.I.1 A *logic error* is a mistake in the program that allows a program to run but causes it to behave incorrectly or unexpectedly.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☑	☑	☐	☐	☐	☑	☐	☐	☐

CRD-2.I.2 A *syntax error* is a mistake in the program where the rules of the programming language are not followed.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☑	☐	☐	☐	☐	☐	☐	☐	☐

CRD-2.I.3 A *run-time error* is a mistake in the program that occurs during the execution of a program. Programming languages define their own run-time errors.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☑	☐	☐	☐	☐	☐	☐	☐	☐

CRD-2.I.4 An *overflow error* is an error that occurs when the computer attempts to handle a number that is outside of the defined range of values can be represented.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	☑	☑	□	□	□	□	□	□	□

CRD-2.I.5 The following are effective ways to find and correct errors:

- test-cases
- hand-tracing
- visualizations
- debuggers
- adding extra output statement(s)

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	☑	☑	□	□	□	□	☑	□	□	□

CRD-2.J Identify inputs and corresponding expected output or behaviors that can be used to check the correctness of a program.

CRD-2.J.1 In the development process, *testing* uses defined inputs to ensure that an algorithm or program is producing the expected outcomes. Programmers use the results from testing to revise their algorithms or programs.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	□	☑	□	☑	□	□	☑	□	□	□

CRD-2.J.2 Defined inputs used to test a program should demonstrate the different expected outcomes that are at or just beyond the extremes (minimum and maximum) of input data.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	□	□	☑	☑	☑	□	□	☑	□	□	□

CRD-2.J.3 Program requirements are needed to identify appropriate defined inputs for testing.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	☑	☑	☑	□	☑	□	□	☑	□	□	□

D8 Data

The way a computer represents data internally is different from the way the data is interpreted and displayed for the user. Programs are used to translate data into a representation more easily understood by people. (EU)

Programs can be used to process data, which allows users to discover information and create new knowledge. (EU)

DAT-1.A Explain how data can be represented using bits.

DAT-1.A.1 Data values can be stored in variables, lists of items, or standalone constants and can be passed as input to (or output from) procedures.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	□	□	□	□	☑	□	□	□	□	□

DAT-1.A.2 Computing devices represent data digitally, which means that the lowest level components of any value are bits.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	☑	□	□	□	□	□	□	□

DAT-1.A.3 *Bit* is shorthand for *binary digit* and is either 0 or 1.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☐	☑	☐	☑	☐	☐	☐	☐	☐

DAT-1.A.4 A *byte* is 8 bits.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☐	☑	☐	☑	☐	☐	☐	☐	☐

DAT-1.A.5 *Abstraction* is the process of reducing complexity by focusing on the main idea. By hiding details irrelevant to the question at hand and bringing together related and useful details, abstraction reduces complexity and allows one to focus on the idea.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☐	☑	☐	☑	☐	☐	☐	☐	☐

DAT-1.A.6 Bits are grouped to represent abstractions. These abstractions include, but are not limited to, numbers, characters, and color.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☐	☑	☐	☑	☐	☐	☐	☐	☐

DAT-1.A.7 The same sequence of bits may represent different types of data in different contexts.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☐	☑	☐	☐	☐	☐	☐	☐	☐

DAT-1.A.8 Analog data have values that change smoothly, rather than in discrete intervals, over time. Some examples of analog data include pitch and volume of music, colors of a painting, or position of a sprinter during a race.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☐	☐	☐	☑	☐	☐	☐	☐	☐

DAT-1.A.9 The use of digital data to approximate real-world analog data is an example of abstraction.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☐	☐	☐	☑	☐	☐	☐	☐	☐

DAT-1.A.10 Analog data can be closely approximated digitally using a *sampling technique*, which means measuring values of the analog signal at regular intervals called *samples*. The samples are measured to figure out the exact bits required to store each sample.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☐	☐	☐	☑	☐	☐	☐	☐	☐

DAT-1.B Explain the consequences of using bits to represent data.

DAT-1.B.1 In many programming languages, the fixed number of bits used to represent integers limits the range of integer values and mathematical operations on those values; this limitation can result in overflow or other errors.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐

DAT-1.B.2 Other programming languages provide an abstraction through which the size of representable integers is limited only by the size of the computer's memory; this is the case for the language defined in the exam reference sheet.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐

DAT-1.B.3 In programming languages, the fixed number of bits used to represent real numbers limits the range and mathematical operations on these values; this limitation can result in round-off and other errors. Some real numbers are represented as approximations in computer storage.
EXCLUSION STATEMENT (EK DAT-1.B.3):
Specific range limitations for real numbers are outside the scope of this course and the AP Exam.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	□	□	□	□	□	□	□	□

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.

DAT-1.C.1 Number bases, including binary and decimal, are used to represent data.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	☑	□	☑	□	□	□	□	□

DAT-1.C.2 Binary (base 2) uses only combinations of the digits zero and one.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	☑	□	☑	□	□	□	□	□

DAT-1.C.3 Decimal (base 10) uses only combinations of the digits 0 – 9.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	☑	□	☑	□	□	□	□	□

DAT-1.C.4 As with decimal, a digit’s position in the binary sequence determines its numeric value. The numeric value is equal to the bit’s value (0 or 1) multiplied by the place value of its position.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	☑	□	☑	□	□	□	□	□

DAT-1.C.5 The place value of each position is determined by the base raised to the power of the position. Positions are numbered starting at the rightmost position with 0 and increasing by 1 for each subsequent position to the left.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	☑	□	☑	□	□	□	□	□

DAT-1.D Compare data compression algorithms to determine which is best in a particular context.

DAT-1.D.1 *Data compression* can reduce the size (number of bits) of transmitted or stored data.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	☑	□	□	□	□	□	□	□

DAT-1.D.2 Fewer bits does not necessarily mean less information.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	☑	□	□	□	□	□	□	□

DAT-1.D.3 The amount of size reduction from compression depends on both the amount of redundancy in the original data representation and the compression algorithm applied.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	☑	□	□	□	□	□	□	□

DAT-1.D.4 *Lossless* data compression algorithms can usually reduce the number of bits stored or transmitted while guaranteeing complete reconstruction of the original data.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☐	☑	☐	☑	☐	☐	☐	☐	☐

DAT-1.D.5 *Lossy* data compression algorithms can significantly reduce the number of bits stored or transmitted but only allow reconstruction of an approximation of the original data.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☐	☑	☐	☐	☐	☐	☐	☐	☐

DAT-1.D.6 *Lossy* data compression algorithms can usually reduce the number of bits stored or transmitted more than lossless compression.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☐	☑	☐	☐	☐	☐	☐	☐	☐

DAT-1.D.7 In situations where quality or ability to reconstruct the original is maximally important, lossless compression algorithms are typically chosen.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☐	☑	☐	☐	☐	☐	☐	☐	☐

DAT-1.D.8 In situations where minimizing data size or short transmission time is maximally important, lossy compression algorithms are typically chosen.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☐	☑	☐	☐	☐	☐	☐	☐	☐

DAT-2.A Describe what information can be extracted from data.

DAT-2.A.1 *Information* is the collection of facts and patterns extracted from data.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☐	☑	☐	☑	☑	☑	☑	☑	☐

DAT-2.A.2 Data provide opportunities for identifying trends, making connections, and addressing problems.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☐	☑	☐	☑	☑	☑	☑	☑	☐

DAT-2.A.3 Digitally processed data may show correlation between variables. A correlation found in data does not necessarily indicate that a causal relationship exists. Additional research is needed to understand the exact nature of the relationship.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☐	☑	☐	☑	☑	☑	☑	☑	☐

DAT-2.A.4 Often, a single source does not contain the data needed to draw a conclusion. It may be necessary to combine data from a variety of sources to formulate a conclusion.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☐	☑	☐	☑	☑	☑	☑	☑	☐

DAT-2.B Describe what information can be extracted from metadata.

DAT-2.B.1 *Metadata* are data about data. For example, the piece of *data* may be an image, while the *metadata* may include the date of creation or the file size of the image.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☐	☑	☐	☐	☐	☐	☐	☐	☐

DAT-2.B.2 Changes and deletions made to metadata do not change the primary data.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☐	☐	☐	☐	☑	☐	☐	☐	☐

DAT-2.B.3 Metadata are used for finding, organizing, and managing information.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☐	☑	☐	☑	☑	☑	☐	☐	☐

DAT-2.B.4 Metadata can increase the effective use of data or data sets by providing additional information.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☐	☑	☐	☐	☐	☐	☐	☐	☐

DAT-2.B.5 Metadata allows data to be structured and organized.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☐	☑	☐	☐	☐	☐	☐	☐	☐

DAT-2.C Identify the challenges associated with processing data.

DAT-2.C.1 The ability to process data depends on the capabilities of the users and their tools.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☐	☐	☐	☑	☑	☑	☐	☑	☐

DAT-2.C.2 Data pose challenges regardless of size. Such as:

- the need to clean data
- incomplete data
- invalid data
- the need to combine data sources

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☐	☐	☐	☑	☑	☑	☐	☐	☐

DAT-2.C.3 Depending on how data were collected, they may not be uniform. For example, if users entered data into an open field, the way they choose to abbreviate, spell, or capitalize something may vary from user to user.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☐	☐	☐	☑	☑	☑	☐	☐	☐

DAT-2.C.4 *Cleaning data* is a process that makes the data uniform without changing its meaning (e.g., replacing all equivalent abbreviations, spellings, and capitalizations with the same word).

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☑	☐	☐	☐	☐	☐	☑	☑	☐	☐	☐

DAT-2.C.5 Problems of bias are often created by the type or source of data being collected. Bias is not eliminated by simply collecting more data.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☐	☐	☐	☑	☐	☑	☐	☑	☐

DAT-2.C.6 The size of the data set affects the amount of information that can be extracted from it.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☐	☐	☐	☑	☑	☑	☐	☑	☐

DAT-2.C.7 Large data sets are difficult to process using a single computer and may require parallel systems.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	□	□	□	☑	☑	□	□	□

DAT-2.C.8 Scalability of systems is an important consideration when working with data sets, as the computational capacity of a system affects how data sets can be processed and stored.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	□	□	☑	☑	☑	□	□	□

DAT-2.D Extract information from data using a program.

DAT-2.D.1 Programs can be used to process data to acquire information.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	□	□	☑	☑	☑	□	☑	□

DAT-2.D.2 Tables, diagrams, text, and other visual tools can be used to communicate insight and knowledge gained from data.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	□	□	☑	☑	☑	□	□	□

DAT-2.D.3 Search tools are useful for efficiently finding information.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	□	□	☑	☑	☑	□	□	□

DAT-2.D.4 Data filtering systems are important tools for finding information and recognizing patterns in the data.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	□	□	☑	☑	☑	□	□	□

DAT-2.D.5 Programs such as spreadsheets help efficiently organize and find trends in information.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	□	□	☑	☑	☑	□	□	□

DAT-2.D.6 Some processes that can be used to extract or modify information from data include the following:

- transforming every element of a data set, such as doubling every element in a list, or extracting the parent’s email from every student record
- filtering a data set, such as keeping only the positive numbers from a list, or keeping only students who signed up for band from a record of all the student
- combining or comparing data in some way, such as adding up a list of numbers, or finding the student who has the highest GPA
- visualizing a data set through a chart, graph, or other visual representation

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	□	□	☑	☑	☑	□	□	□

DAT-2.E Explain how programs can be used to gain insight and knowledge from data.

DAT-2.E.1 Programs are used in an iterative and interactive way when processing information to allow users to gain insight and knowledge about data.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	□	□	☑	☑	☑	☑	☑	□

DAT-2.E.2 Programmers can use programs to filter and clean digital data, thereby gaining insight and knowledge.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☐	☐	☐	☑	☑	☑	☐	☐	☐

DAT-2.E.3 Combining data sources, clustering data, and classifying data are parts of the process of using programs to gain insight and knowledge from data.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☐	☐	☐	☑	☑	☑	☐	☑	☐

DAT-2.E.4 Insight and knowledge can be obtained from translating and transforming digitally represented information.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☐	☐	☐	☑	☑	☑	☑	☐	☐

DAT-2.E.5 Patterns can emerge when data is transformed using programs.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☐	☐	☐	☑	☑	☑	☑	☐	☐

D9 Algorithms and Programming

To find specific solutions to generalizable problems, programmers represent and organize data in multiple ways. (EU)

The way statements are sequenced and combined in a program determines the computed result. Programs incorporate iteration and selection constructs to represent repetition and make decisions to handle varied input values. (EU)

Programmers break down problems into smaller and more manageable pieces. By creating procedures and leveraging parameters, programmers generalize processes that can be reused. Procedures allow programmers to draw upon existing code that has already been tested, allowing them to write programs more quickly and with more confidence. (EU)

There exist problems that computers cannot solve, and even when a computer can solve a problem, it may not be able to do so in a reasonable amount of time. (EU)

AAP-1.A Represent a value with a variable.

AAP-1.A.1 A *variable* is an abstraction inside a program that can hold a value. Each variable has associated data storage that represents one value at a time, but that value can be a list or other collection that in turn contains multiple values.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☐	☑	☐	☐	☐	☐	☑	☐	☐	☐	☐

AAP-1.A.2 Using meaningful variable names helps with the readability of program code and understanding of what values are represented by the variables.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	☑	☐	☐	☐	☐	☐	☐	☐	☐	☐

AAP-1.A.3 Some programming languages provide *types* to represent data, which are referenced using variables. These types include numbers, Booleans, lists, and strings.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	☑	☑	☐	☐	☐	☐	☐	☐	☐	☐

AAP-1.A.4 Some values are better suited to representation using one type of data rather than another.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	☑	☑	☑	□	□	□	□	□	□	□	□

AAP-1.B Determine the value of a variable as a result of an assignment.

AAP-1.B.1 The assignment operator allows a program to change the value represented by a variable.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	□	□	□	□	□	□	□	□	□	□

AAP-1.B.2 The exam reference sheet provides the "←" operator to use for assignment. For example,

```
a ← expression
```

evaluates expression and then assigns the result to the variable a.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	☑	□	□	□	□	□	□	□	□	□

AAP-1.B.3 The value stored in a variable will be the most recent value assigned. For example:

```
a ← 1
b ← a
a ← 2
display(b)
```

still displays 1.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	☑	□	□	□	□	□	□	□	□	□

AAP-1.C Represent a list or string using a variable.

AAP-1.C.1 A *list* is an ordered sequence of elements. For example,

```
[value1, value2, value3, …]
```

describes a list where value1 is the first element, value 2 is the second element, value 3 is the third element, and so on.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	☑	□	□	□	□	□	□	□	□	□

AAP-1.C.2 An *element* is an individual value in a list that is assigned a unique index.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	☑	□	□	□	□	□	□	□	□	□

AAP-1.C.3 An *index* is a common method for referencing the elements in a list or string using natural numbers.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	☑	□	□	□	□	□	□	□	□	□

AAP-1.C.4 A *string* is an ordered sequence of characters.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	☑	☑	□	□	□	□	□	□	□	□	□

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.

AAP-1.D.1 Data abstraction provides a separation between the abstract properties of a data type and the concrete details of its representation.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☑	☑	☐	☐	☐	☐	☑	☐	☐	☐	☐

AAP-1.D.2 Data abstractions manage complexity in programs by giving a collection of data a name without referencing the specific details of the representation.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☑	☑	☐	☐	☐	☐	☐	☐	☐	☐	☐

AAP-1.D.3 Data abstractions can be created using lists.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☑	☑	☐	☐	☐	☐	☑	☑	☐	☐	☐

AAP-1.D.4 Developing a data abstraction to implement in a program can result in a program that is easier to develop and maintain.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☑	☑	☐	☐	☐	☐	☑	☑	☐	☐	☐

AAP-1.D.5 Data abstractions often contain different types of elements.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☑	☑	☐	☐	☐	☐	☑	☐	☐	☐	☐

AAP-1.D.6 The use of lists allows multiple related items to be treated as a single value. Lists are referred to by different names, such as *array*, depending on the programming language.
EXCLUSION STATEMENT(EK APP-1.D.6):
The use of linked lists is outside the scope of this course and the AP Exam.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☑	☑	☐	☐	☐	☐	☐	☐	☐	☐	☐

AAP-1.D.7 The exam reference sheet provides the notation

[value1, value2, value3…]

to create a list with those values as the first, second, third, and so on items. For example,

- aList ← [value1, value2, value3,...]
  creates a new list that contains the values value1, value2, value3, and … at indices 1, 2, 3, and … respectively and assigns it to aList.
- aList ← [ ]
  creates a new empty list and assigns it to aList.
- aList ← bList
  assigns a copy of the list bList to the list aList. For example, if bList contains [20, 40, 60], then aList will also contain [20, 40, 60] after the assignment.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	☑	☐	☐	☐	☐	☐	☐	☐	☐	☐

AAP-1.D.8 The exam reference sheet describes a list structure whose index values are 1 through the number of elements in the list, inclusive. For all list operations, if a list index is less than 1 or greater than the length of the list, an error message is produced and the program will terminate.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☑	☑	☑	☐	☐	☐	☐	☐	☐	☐	☐

AAP-2.A Express an algorithm that uses sequencing without using a programming language.

AAP-2.A.1 An *algorithm* is a finite set of instructions that accomplish a specific task.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	☑	□	□	□	□	□	□	□	☑	□

AAP-2.A.2 Beyond visual and textual programming languages, algorithms can be expressed in a variety of ways, such as natural language, diagrams, and pseudocode.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	☑	□	□	□	□	□	□	□	□	□

AAP-2.A.3 Algorithms executed by a program are implemented using programming languages.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	☑	□	□	□	□	□	□	□	□	□

AAP-2.A.4 Every algorithm can be constructed using combinations of sequencing, selection, and iteration.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	☑	□	□	□	□	□	□	□	□	□

AAP-2.B Represent a step-by-step algorithmic process using sequential code statements.

AAP-2.B.1 *Sequencing* is the application of each step of an algorithm in the order in which the code statements are given.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	□	□	□	□	□	□	□	□	□	□

AAP-2.B.2 A *code statement* is a part of program code that expresses an action to be carried out.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	☑	□	□	□	□	□	□	□	□	□

AAP-2.B.3 An *expression* can consist of a value, a variable, an operator, or a procedure call that returns a value.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	☑	☑	□	□	□	□	□	□	□	□	□

AAP-2.B.4 Expressions are evaluated to produce a single value.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	☑	□	□	□	□	□	□	□	□	□

AAP-2.B.5 The evaluation of expressions follows a set order of operations defined by the programming language.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	☑	□	□	□	□	□	□	□	□	□

AAP-2.B.6 Sequential statements execute in the order they appear in the code segment.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	☑	□	□	□	□	□	□	□	□	□

AAP-2.B.7 Clarity and readability are important considerations when expressing an algorithm in a programming language.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	☑	□	□	□	□	□	□	□	□	□

AAP-2.C Evaluate expressions that use arithmetic operators.

AAP-2.C.1 Arithmetic operators are part of most programming languages and include addition, subtraction, multiplication, division, and modulus operators.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	☑	□	□	□	□	□	□	□	□	□

AAP-2.C.2 The exam reference sheet provides a MOD b, which evaluates to the remainder when a is divided by b. Assume that a is an integer greater than or equal to 0 and b is an integer greater than 0. For example, 17 MOD 5 evaluates to 2.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	☑	□	□	□	□	□	□	□	□	□

AAP-2.C.3 The exam reference sheet provides the arithmetic operators +, -, *, /, and MOD.

- a + b
- a – b
- a* b
- a / b
- a MOD b

These are used to perform arithmetic on a and b. For example, 17 / 5 evaluates to 3.4.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	□	☑	□	□	□	□	□	□	□	□	□

AAP-2.C.4 The order of operations used in mathematics applies when evaluating expressions. The MOD operator has the same precedence as the * and / operators.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	□	☑	□	□	□	□	□	□	□	□	□

AAP-2.D Evaluate expressions that manipulate strings.

AAP-2.D.1 *String concatenation* joins together two or more strings end-to-end to make a new string.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	☑	☑	□	□	□	□	□	□	□	□	□

AAP-2.D.2 A *substring* is part of an existing string.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	□	□	□	□	□	□	□	□

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.E.1 A *Boolean value* is either true or false.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	☑	☑	□	□	□	□	□	□	□	□	□

AAP-2.E.2 The exam reference sheet provides the following relational operators =, ≠, >,<, ≥, and ≤.

- a = b
- a ≠ b
- a > b
- a < b
- a ≥ b
- a ≤ b

These are used to test the relationship between two variables, expressions, or values. A comparison using a relational operator evaluates to a Boolean value. For example, a = b evaluates to true if a and b are equal; otherwise, it evaluates to false.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	☑	☑	□	□	□	□	□	□	□	□	□

AAP-2.F For relationships between Boolean values: (a) Write expressions using logical operators. (b) Evaluate expressions that use logic operators.

AAP-2.F.1 The exam reference sheet provides the logical operators NOT, AND, and OR, which evaluate to a Boolean value.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	☑	☑	□	□	□	□	□	□	□	□	□

AAP-2.F.2 The exam reference sheet provides the NOT condition, which evaluates to true if condition is false; otherwise it evaluates to false.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	☑	☑	□	□	□	□	□	□	□	□	□

AAP-2.F.3 The exam reference sheet provides condition1 AND condition2, which evaluates to true if both condition1 and condition2 are true; otherwise it evaluates to false.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	☑	☑	□	□	□	□	□	□	□	□	□

AAP-2.F.4 The exam reference sheet provides condition1 OR condition2, which evaluates to true if condition1 is true or if condition2 is true or if both condition1 and condition2 are true; otherwise it evaluates to false.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	☑	☑	□	□	□	□	□	□	□	□	□

AAP-2.F.5 The operand for a logical operator is either a Boolean expression or a single Boolean value.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	☑	☑	□	□	□	□	□	□	□	□	□

AAP-2.G Express an algorithm that uses selection without using a programming language.

AAP-2.G.1 *Selection* determines which parts of an algorithm are executed based on a condition being true or false.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	☑	☑	□	□	□	□	□	□	□	□	□

AAP-2.H For selection: (a) Write conditional statements. (b) Determine the result of conditional statements.

AAP-2.H.1 Conditional statements or “if-statements” affect the sequential flow of control by executing different statements based on the value of a Boolean expression.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	☑	□	□	□	□	□	□	□	□	□

AAP-2.H.2 The exam reference sheet provides

```
IF(condition)
{
 <block of statements>
}
```

in which the code in block of statements is executed if the Boolean expression condition evaluates to true; no action is taken if condition evaluates to false.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	□	☑	□	□	□	□	□	□	□	□	□

AAP-2.H.3 The exam reference sheet provides

```
IF(condition)
{
 <first block of statements>
}
ELSE
{
 <second block of statements>
}
```

in which the code in first block of statements is executed if the Boolean expression condition evaluates to true; otherwise, the code in second block of statements is executed.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	□	☑	□	□	□	□	□	□	□	□	□

AAP-2.I For nested selection: (a) Write nested conditional statements. (b) Determine the result of nested conditional statements.

AAP-2.I.1 Nested conditional statements or “else if” statements consist of conditional statements within conditional statements.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	□	☑	□	□	□	□	□	□	□	□	□

AAP-2.I.2 If the Boolean condition of the initial conditional statement evaluates to false, then the Boolean condition of the nested conditional statement is evaluated.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	□	□	□	□	□	□	□	□	□	□	□

AAP-2.J Express an algorithm that uses iteration without using a programming language.

AAP-2.J.1 *Iteration* is a repeating portion of an algorithm. Iteration repeats a specified number of times or until a given condition is met.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	□	□	☑	□	□	□	□	□	□	□	□

AAP-2.K For iteration: (a) Write iteration statements. (b) Determine the result or side effect of iteration statements.

AAP-2.K.1 Iteration statements change the sequential flow of control by repeating a set of statements zero or more times, until a stopping condition is met.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	☑	☐	☐	☐	☐	☐	☐	☐	☐	☐

AAP-2.K.2 The exam reference sheet provides

```
REPEAT n TIMES
{
 <block of statements>
}
```

in which the block of statements is executed n times.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☐	☑	☐	☐	☐	☐	☐	☐	☐	☐	☐

AAP-2.K.3 The exam reference sheet provides

```
REPEAT UNTIL(condition)
{
 <block of statements>
}
```

in which the code in block of statements is repeated until the Boolean expression condition evaluates to true.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☐	☑	☐	☐	☐	☐	☐	☐	☐	☐	☐

AAP-2.K.4 In REPEAT UNTIL(condition) iteration, an infinite loop occurs when the ending condition will never evaluate to true.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☐	☑	☐	☐	☐	☐	☐	☐	☐	☐	☐

AAP-2.K.5 In REPEAT UNTIL(condition) iteration, if the conditional evaluates to true initially, the loop body is not executed at all, due to the condition being checked before the loop.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☐	☑	☐	☐	☐	☐	☐	☐	☐	☐	☐

AAP-2.L Compare multiple algorithms to determine if they yield the same side effect or result.

AAP-2.L.1 Algorithms can be written in different ways and still accomplish the same tasks.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	☑	☐	☐	☐	☐	☐	☐	☐	☐	☐

AAP-2.L.2 Algorithms that appear similar can yield different side effects or results.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☐	☑	☐	☐	☐	☐	☐	☐	☐	☐	☐

AAP-2.L.3 Some selections can be written as equivalent Boolean expressions.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☑	☐	☐	☐	☐	☐	☐	☐	☐

AAP-2.L.4 Some Boolean expressions can be written as equivalent selections.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	☑	□	□	□	□	□	□	□	□

AAP-2.L.5 Different algorithms can be developed or used to solve the same problem.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	□	☑	□	□	□	□	□	□	□	□	□

AAP-2.M For algorithms: (a) Create algorithms. (b) Combine and modify existing algorithms.

AAP-2.M.1 Algorithms can be created from an idea, by combining existing algorithms, or by modifying existing algorithms.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	☑	☑	□	□	□	□	□	□	□	□	□

AAP-2.M.2 Knowledge of existing algorithms can help in constructing algorithms. Some existing algorithms include:

- determining the maximum or minimum value of 2 or more numbers
- computing the sum or average of 2 or more numbers
- identifying if an integer is or is not evenly divisible by another integer
- determining a robot’s path through a maze

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	☑	□	□	□	□	□	□	□	□	□

AAP-2.M.3 Using existing correct algorithms as building blocks for constructing another algorithm has benefits such as reducing development time, reducing testing, and simplifying the identification of errors.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	☑	□	□	□	□	□	□	□	□	□

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.

The exam reference sheet provides basic operations on lists including:

AAP-2.N.1 • accessing an element by index aList[i] accesses the element of aList at index i. The first element of aList is at index 1 and accessed using the notation aList[1].

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	☑	□	□	□	□	□	□	□	□	□

AAP-2.N.1 • assigning a value of an element of a list to a variable x ← aList [i] assigns the value of aList[i] to the variable x.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	☑	□	□	□	□	□	□	□	□	□

AAP-2.N.1 • assigning a value to an element of a list;

- aList[i] ← x - assigns the value of x to aList[i].
- aList[i] ← aList[j] - assigns the value of aList[j] to aList[i].

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	☑	□	□	□	□	□	□	□	□	□

AAP-2.N.1 • inserting elements at a given index; INSERT(aList, i, value) - shifts to the right any values in aList at indices greater than or equal to i. The length of the list is increased by 1, and value is placed at index i in aList.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	☑	□	□	□	□	□	□	□	□	□

AAP-2.N.1 • adding elements to the end of the list APPEND(aList, value) increases the length of aList by 1, and value is placed at the end of aList.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	☑	□	□	□	□	□	□	□	□	□

AAP-2.N.1 • removing elements REMOVE(aList, i) removes the item at index i in aList and shifts to the left any values at indices greater than i. The length of aList is decreased by 1.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	☑	□	□	□	□	□	□	□	□	□

AAP-2.N.1 • determining the length of a list; LENGTH(aList) Evaluates to the number of elements currently in aList.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	☑	☑	□	□	□	□	□	□	□	□	□

AAP-2.N.2 List procedures are implemented in accordance with the syntax rules of the language.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	☑	□	□	□	□	□	□	□	□	□	□

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.O.1 Traversing a list can be a complete traversal, where all elements in the list are accessed, or a partial traversal, where only a portion of elements are accessed.
EXCLUSION STATEMENT (EK AAP-2.O.1):
Traversing multiple lists at the same time using the same index for both (parallel traversals) is outside the scope of this course and the AP Exam.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	□	□	□	□	□	□	□	□	□	□

AAP-2.O.2 Iteration statements can be used to traverse a list.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	□	□	□	□	□	□	□	□	□	□

AAP-2.O.3 The exam reference sheet provides

```
FOR EACH item IN aList
{
 <block of statements>
}
```

The variable item is assigned the value of each element of aList sequentially, in order, from the first element to the last element. The code in block of statements is executed once for each assignment of item.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	□	□	□	□	□	□	□	□	□	□	□

AAP-2.O.4 Knowledge of existing algorithms that use iteration can help in constructing new algorithms. Some examples of existing algorithms that are often used with lists include:

- determining a minimum or maximum value in a list
- computing a sum or average

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☑	☐	☐	☐	☐	☐	☑	☑	☐	☐	☐

AAP-2.O.5 Linear search or sequential search algorithms check each element of a list, in order, until the desired value is found or all elements in the list have been checked.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐

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.

AAP-2.P.1 The binary search algorithm starts at the middle of a sorted data set of numbers and eliminates half of the data; this process repeats until the desired value is found or all elements have been eliminated.

EXCLUSION STATEMENT (EK: AAP-2.P.1):

Specific implementations of the binary search are outside the scope of the course and the AP Exam.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☐	☐	☐	☑	☐	☐	☐	☐	☐

AAP-2.P.2 Data must be in sorted order to use the binary search algorithm.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☐	☐	☐	☑	☐	☐	☐	☐	☐

AAP-2.P.3 Binary search is often more efficient than sequential / linear search when the data is sorted.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☐	☐	☐	☑	☐	☐	☐	☐	☐

AAP-3.A Determine the result or effect of a procedure call.

AAP-3.A.1 A *procedure* is a named group of programming instructions that may have parameters and return values.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☑	☑	☐	☐	☐	☐	☐	☐	☐	☐	☐

AAP-3.A.2 Procedures are referred to by different names, such as *method* or *function*, depending on the programming language.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☑	☑	☐	☐	☐	☐	☐	☐	☐	☐	☐

AAP-3.A.3 Parameters are input variables of a procedure. Arguments specify the values of the parameters when a procedure is called.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☑	☑	☐	☐	☐	☐	☐	☐	☐	☐	☐

AAP-3.A.4 A procedure call interrupts the sequential execution of statements, causing the program to execute the statements within the procedure before continuing. Once the last statement in the procedure (or a return statement) has executed, flow of control is returned to the point immediately following where the procedure was called.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☑	☑	☐	☐	☐	☐	☐	☐	☐	☐	☐

AAP-3.A.5 The exam reference sheet provides

```
procName (arg1, arg2, …)
```

as a way to call

```
PROCEDURE procName(parameter1, parameter 2, …)
```

which takes zero or more arguments; arg1 is assigned to parameter1, arg2 is assigned to parameter2, and so on.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☑	☑	☐	☐	☐	☐	☐	☐	☐	☐	☐

AAP-3.A.6 The exam reference sheet provides the procedure

```
DISPLAY(expression)
```

to display the value of expression, followed by a space.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☑	☑	☐	☐	☐	☐	☐	☐	☐	☐	☐

AAP-3.A.7 The exam reference sheet provides the

```
RETURN(expression)
```

statement, which is used to return the flow of control to the point where the procedure was called and to return the value of expression.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☑	☑	☐	☐	☐	☐	☐	☐	☐	☐	☐

AAP-3.A.8 The exam reference sheet provides

```
result ← procName(arg1, arg2,)
```

to assign to result the "value of the procedure" being returned by calling

```
PROCEDURE procName(parameter1,parameter2,)
{
   <block of statements>
   RETURN(expression)
}
```

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☑	☑	☐	☐	☐	☐	☐	☐	☐	☐	☐

AAP-3.A.9 The exam reference sheet provides procedure

```
INPUT()
```

which accepts a value from the user and is equivalent to the input value.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☑	☑	☐	☐	☐	☐	☐	☐	☐	☐	☐

AAP-3.B Explain how the use of procedural abstraction manages complexity in a program.

AAP-3.B.1 One common type of abstraction is procedural abstraction which provides a name for a process and allows a procedure to be used only knowing what it does, not how it does it.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☑	☑	☐	☐	☐	☐	☐	☐	☐	☐	☐

AAP-3.B.2 Procedural abstraction allows a solution to a large problem to be based on the solution of smaller subproblems. This is accomplished by creating procedures to solve each of the subproblems.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	☑	☑	□	□	□	□	□	□	□	□	□

AAP-3.B.3 The subdivision of a computer program into separate subprograms is called *modularity*.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	☑	☑	□	□	□	□	□	☑	□	□	□

AAP-3.B.4 A procedural abstraction may extract shared features to generalize functionality instead of duplicating code. This allows for program code reuse, which helps to manage complexity.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	☑	☑	□	□	□	□	□	□	□	□	□

AAP-3.B.5 Using parameters allows procedures to be generalized, enabling the procedures to be reused with a range of input values or arguments.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	☑	□	□	□	□	□	□	☑	□	□	□

AAP-3.B.6 Using procedural abstraction helps improve code readability.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	☑	☑	□	□	□	□	□	□	□	□	□

AAP-3.B.7 Using procedural abstraction in a program allows programmers to change the internals of the procedure (to make it faster, more efficient, use less storage, etc.) without needing to notify users of the change as long as what the procedure does is preserved.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	☑	☑	□	□	□	□	□	□	□	□	□

AAP-3.C Develop procedural abstractions to manage complexity in a program by writing procedures.

AAP-3.C.1 The exam reference sheet provides

```
PROCEDURE procName(parameter1, parameter2, …)
{
 <block of statements>
}
```

which is used to define a procedure that takes zero or more arguments. The procedure contains block of statements.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	☑	□	□	□	□	□	□	□	□	□	□

AAP-3.C.2 The exam reference sheet provides

```
PROCEDURE procName(parameter1, parameter2, …)
{
 <block of statements>
 RETURN(expression)
}
```

which is used to define a procedure that takes zero or more arguments. The procedure contains block of statements and returns the value of expression. The RETURN statement may appear at any point inside the procedure and causes an immediate return from the procedure back to the calling statement.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐

AAP-3.D Select appropriate libraries or existing code segments to use in creating new programs.

AAP-3.D.1 A software library contains procedures that may be used in creating new programs.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☑	☑	☑	☐	☐	☐	☐	☐	☐	☐	☐

AAP-3.D.2 Existing code segments can come from internal or external sources, such as libraries previously written code.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☑	☑	☐	☑	☐	☑	☑	☑	☑	☐

AAP-3.D.3 The use of libraries simplifies the task of creating complex programs.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☑	☑	☑	☑	☐	☑	☑	☑	☑	☐

AAP-3.D.4 Application program interfaces (APIs) are specifications for how the procedures in a library behave and can be used.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☑	☐	☑	☐	☑	☐	☑	☐	☐

AAP-3.D.5 Documentation for an API/library is necessary in understanding the behavior(s) provided by the API/library and how to use them.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☑	☐	☑	☐	☑	☐	☑	☐	☐

AAP-3.E For generating random values: (a) Write expressions to generate possible values. (b) Evaluate expressions to determine the possible results.

AAP-3.E.1 The exam reference sheet provides

```
RANDOM(a, b)
```

which generates and returns a random integer from a to b, inclusive. Each result is equally likely to occur. For example, RANDOM(1, 3) could return 1, 2, or 3.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	☑	☐	☐	☐	☐	☐	☐	☐	☐	☐

AAP-3.E.2 Using random number generation in a program means each execution could produce a different result.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☑	☑	☐	☐	☐	☐	☑	☐	☑	☐	☐

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-3.F.1 Simulations are abstractions of more complex objects or phenomena for a specific purpose.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	□	□	□	□	□	☑	□	□

AAP-3.F.2 A *simulation* is a representation that uses varying sets of values to reflect the changing state of the phenomenon.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	□	□	□	□	□	☑	□	□

AAP-3.F.3 Simulations often mimic real-world events with the purpose of drawing inferences allowing investigation of the phenomenon without the constraints of the real world.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	□	□	□	□	□	☑	□	□

AAP-3.F.4 The process of developing an abstract simulation involves removing specific details or simplifying functionality.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	□	□	□	□	□	☑	□	□

AAP-3.F.5 Simulations can contain bias derived from the choices of real-world elements that were included or excluded.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	□	□	□	□	□	☑	□	□

AAP-3.F.6 Simulations are most useful when real-world events are impractical for experiments (i.e., too big, too small, too fast, too slow, too expensive, or too dangerous).

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	□	□	□	□	□	☑	□	□

AAP-3.F.7 Simulations facilitate the formulation and refinement of hypotheses related to the objects or phenomena under consideration.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	□	□	□	□	□	☑	□	□

AAP-3.F.8 Random number generators can be used to simulate the variability that exists in the real world.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	□	□	□	□	□	☑	□	□

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.

AAP-4.A.1 A *problem* is a general description of a task that can (or cannot) be solved algorithmically. An *instance* of a problem also includes specific input. For example, sorting is a problem; sorting the list (2,3,1,7) is an instance of the problem.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	□	□	□	☑	□	□	□	□

AAP-4.A.2 A *decision problem* is a problem with a yes/no answer (e.g., is there a path from A to B?). An *optimization problem* is a problem with the goal of finding the "best" solution among many (e.g., what is the shortest path from A to B?).

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	□	□	□	□	□	☑	□	□

AAP-4.A.3 *Efficiency* is an estimation of the amount of computational resources used by an algorithm. Efficiency is typically expressed as a function of the size of the input.
EXCLUSION STATEMENT (EK AAP-4.A.3):
Formal analysis of algorithms (Big-O) and formal reasoning using mathematical formulas are outside the scope of this course and the AP Exam.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	□	□	□	☑	□	□	□	□

AAP-4.A.4 An algorithm’s efficiency is determined through formal or mathematical reasoning.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	□	□	□	☑	□	□	□	□

AAP-4.A.5 An algorithm's efficiency can be informally measured by determining the number of times a statement or group of statements executes.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	□	□	□	☑	□	□	□	□

AAP-4.A.6 Different correct algorithms for the same problem can have different efficiencies.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	□	□	□	☑	□	□	□	□

AAP-4.A.7 Algorithms with a polynomial efficiency or slower (constant, linear, square, cube, etc.) are said to run in a *reasonable amount of time*. Algorithms with exponential or factorial efficiencies are examples of algorithms that run in an *unreasonable amount of time*.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	□	□	□	☑	□	□	□	□

AAP-4.A.8 Some problems cannot be solved in a reasonable amount of time because there is no efficient algorithm for solving them. In these cases, approximate solutions are sought.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	□	□	□	□	□	☑	□	□

AAP-4.A.9 A *heuristic* is an approach to a problem that produces a solution that is not guaranteed to be optimal but may be used when techniques that are guaranteed to always find an optimal solution are impractical.
EXCLUSION STATEMENT (AAP-4.A.9):
Specific heuristic solutions are outside the scope of this course and the AP Exam.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	□	□	□	□	□	☑	□	□

AAP-4.B Explain the existence of undecidable problems in computer science.

AAP-4.B.1 A decidable problem is a decision problem for which an algorithm can be written to produce a correct output for all inputs (e.g., “Is the number even?”).

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	□	□	□	□	□	□	□	□

AAP-4.B.2 An *undecidable problem* is one for which no algorithm can be constructed that is always capable of providing a correct yes-or-no answer.
EXCLUSION STATEMENT (EK AAP-4.B.2):
Determining whether a given problem is undecidable is outside the scope of this course and the AP Exam.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐

AAP-4.B.3 An undecidable problem may have some instances that have an algorithmic solution, but there is no algorithmic solution that solves all instances of the problem.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐

D10 Computing Systems and Networks

There exist problems that the computer cannot solve, and even when the computer can solve a problem, it may not be able to do so in a reasonable amount of time.

Computer systems and networks facilitate how data are transferred. (EU)

Parallel and distributed computing leverage multiple computers to more quickly solve complex problems or process large data sets. (EU)

CSN-1.A Explain how computing devices work together in a network.

CSN-1.A.1 A computing device is a physical artifact that can run a program. Some examples include computers, tablets, servers, routers, and smart sensors.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☐	☑	☐	☐	☐	☐	☐	☐	☐

CSN-1.A.2 A *computing system* is a group of computing devices and programs working together for a common purpose.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☐	☑	☐	☐	☐	☐	☐	☐	☐

CSN-1.A.3 A *computer network* is a group of interconnected computing devices capable of sending or receiving data.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☐	☑	☐	☐	☐	☐	☐	☐	☐

CSN-1.A.4 A computer network is a type of a computing system.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☐	☑	☐	☐	☐	☐	☐	☐	☐

CSN-1.A.5 A *path* between two computing devices on a computer network (a sender and a receiver) is a sequence of directly connected computing devices that begins at the sender and ends at the receiver.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☐	☑	☐	☐	☐	☐	☐	☐	☐

CSN-1.A.6 *Routing* is the process of finding a path from sender to receiver.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☐	☑	☐	☐	☐	☐	☐	☑	☐

CSN-1.A.7 The *bandwidth* of a computer network is the maximum amount of data that can be sent in a fixed amount of time.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	☑	□	□	□	□	□	□	□

CSN-1.A.8 Bandwidth is usually measured in bits per second.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	☑	□	□	□	□	□	□	□

CSN-1.B Explain how the Internet works.

CSN-1.B.1 The Internet is a computer network consisting of interconnected networks that use standardized, open (nonproprietary) communication protocols.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	☑	☑	□	□	□	□	☑	□

CSN-1.B.2 Access to the Internet depends on the ability to connect a computing device to an Internet-connected device.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	☑	☑	□	□	□	□	☑	□

CSN-1.B.3 A *protocol* is an agreed-upon set of rules that specify the behavior of some system.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	☑	☑	□	□	□	□	□	□

CSN-1.B.4 The protocols used in the Internet are *open*, which allows users to easily connect additional computing devices to the Internet.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	☑	☑	□	□	□	□	□	□

CSN-1.B.5 Routing on the internet is usually dynamic; it is not specified in advance.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	☑	☑	□	□	□	□	☑	□

CSN-1.B.6 The *scalability* of a system is the capacity for the system to change in size and scale to meet new demands.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	☑	☑	□	□	□	□	☑	□

CSN-1.B.7 The Internet was designed to be scalable.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	☑	☑	□	□	□	□	☑	□

CSN-1.C Explain how data are sent through the Internet via packets.

CSN-1.C.1 Information is passed through the Internet as a *data stream*. Data streams contain chunks of data, which are encapsulated in *packets*.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	☑	□	□	□	□	□	□	□

CSN-1.C.2 Packets contain a chunk of data and metadata used for routing the packet between the origin and the destination on the Internet, as well as for data reassembly.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	☑	□	□	□	□	□	□	□

CSN-1.C.3 Packets may arrive at the destination in order, out of order, or not at all.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	☑	□	□	□	□	□	□	□

CSN-1.C.4 IP, TCP, and UDP are common protocols used on the Internet.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	☑	□	□	□	□	□	□	□

CSN-1.D Describe the differences between the Internet and the World Wide Web.

CSN-1.D.1 The World Wide Web is a system of linked pages, programs, and files.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	☑	□	□	□	□	□	□	□

CSN-1.D.2 The HTTP protocol is the used on the World Wide Web.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	☑	□	□	□	□	□	□	□

CSN-1.D.3 The World Wide Web uses the Internet.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	☑	□	□	□	□	□	□	□

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-1.E.1 The internet has been engineered to be fault-tolerant, with abstractions for routing and transmitting data.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	☑	□	□	□	□	□	☑	□

CSN-1.E.2 *Redundancy* is the inclusion of extra components that can be used to mitigate failure of the system if other components fail.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	☑	□	□	□	□	□	☑	□

CSN-1.E.3 One way to accomplish network redundancy is by having more than one path between any two connected devices.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	☑	□	□	□	□	□	☑	□

CSN-1.E.4 If a particular device or connection on the internet fails, subsequent data will be sent via a different route, if possible.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	☑	□	□	□	□	□	☑	□

CSN-1.E.5 When a system can support failures and still continue to function, it is called *fault-tolerant*. This is important because elements of complex systems fail at unexpected times, often in groups, and fault tolerance allows users to continue to use the network.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☐	☑	☐	☐	☐	☐	☐	☑	☐

CSN-1.E.6 Redundancy within a system often requires additional resources but can provide the benefit of fault tolerance.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☐	☑	☐	☐	☐	☐	☐	☑	☐

CSN-1.E.7 The redundancy of routing between two points on the Internet increases the reliability of the Internet and helps it scale to more devices and more people.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☐	☑	☐	☐	☐	☐	☐	☑	☐

CSN-2.A For sequential, parallel, and distributed computing: (a) Compare problem solutions. (b) Determine the efficiency of solutions.

CSN-2.A.1 *Sequential computing* is a computational model in which operations are performed in order one at a time.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☐	☑	☐	☐	☐	☐	☐	☐	☐

CSN-2.A.2 *Parallel computing* is a computational model where the program is broken into multiple smaller sequential computing operations some of which are performed simultaneously.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☐	☑	☐	☑	☐	☐	☐	☐	☐

CSN-2.A.3 *Distributed computing* is a computational model in which multiple devices are used to run a program.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☐	☑	☐	☐	☐	☐	☐	☐	☐

CSN-2.A.4 Comparing efficiency of solutions can be done by comparing the time it takes them to perform the same task.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☐	☑	☐	☐	☐	☐	☐	☐	☐

CSN-2.A.5 A sequential solution takes as long as the sum of all of its steps.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☐	☑	☐	☐	☐	☐	☐	☐	☐

CSN-2.A.6 A parallel computing solution takes as long as its sequential tasks plus the longest of the tasks done in parallel.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☐	☑	☐	☑	☐	☐	☐	☐	☐

CSN-2.A.7 The “speedup” of a parallel solution is measured in the time it took to complete the task sequentially divided by the time it took to complete the task when done in parallel.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☐	☑	☐	☐	☐	☐	☐	☐	☐

CSN-2.B Describe benefits and challenges of parallel and distributed computing.

CSN-2.B.1 Parallel computing consists of a parallel portion and a sequential portion.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	☑	□	□	□	□	□	□	□

CSN-2.B.2 Solutions that use parallel computing can scale more effectively than solutions that use sequential computing.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	☑	□	□	□	□	□	□	□

CSN-2.B.3 Distributed computing allows problems to be solved that could not be solved on a single computer, either because of the processing time or storage needs involved.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	☑	□	□	□	□	□	□	□

CSN-2.B.4 Distributed computing allows much larger problems to be solved quicker than they could be solved using a single computer.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	☑	□	□	□	□	□	□	□

CSN-2.B.5 When increasing the use of parallel computing in a solution, the efficiency of the solution is still limited by the sequential portion. This means at some point, adding parallel portions will no longer increase efficiency.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	☑	□	☑	□	□	□	□	□

D11 Impact of Computing

While computing innovations are typically designed to achieve a specific purpose, they may have unintended consequences. (EU)

The use of computing innovations may involve risks to your personal safety and identity. (EU)

IOC-1.A Explain how an effect of a computing innovation can be both beneficial and harmful.

IOC-1.A.1 People are creators of computing innovations.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	☑	□	□	□	□	□	□	☑

IOC-1.A.2 The way people complete tasks often changes to incorporate new computing innovations.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	☑	□	□	□	□	□	☑	☑

IOC-1.A.3 Not every effect of a computing innovation is anticipated in advance.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	☑	□	□	□	□	□	☑	☑

IOC-1.A.4 A single effect can be viewed as both beneficial and harmful by different people, or even by the same person.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	☑	□	□	□	□	□	☑	☑

IOC-1.A.5 Advances in computing have generated and increased creativity in other fields, such as medicine, engineering, communications, and the arts.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	☑	□	□	□	□	□	☑	☑

IOC-1.B Explain how a computing innovation can have an impact beyond its intended purpose.

IOC-1.B.1 Computing innovations can be used in ways that their creators had not originally intended:

- The World Wide Web was originally intended only for rapid and easy exchange of information within the scientific community.
- Targeted advertising is used to help businesses, but it can be misused at both individual and aggregate levels.
- Machine learning and data mining have enabled innovation in medicine, business, and science, but information discovered in this way has also been used to discriminate against groups of individuals.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	☑	□	□	□	□	□	☑	☑

IOC-1.B.2 Some of the ways computing innovations can be used may have a harmful impact on society, economy, or culture.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	☑	□	□	□	□	□	☑	☑

IOC-1.B.3 Responsible programmers try to consider the unintended ways their computing innovations can be used and the potential beneficial and harmful effects of these new uses.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	☑	□	□	□	□	□	□	☑

IOC-1.B.4 It is not possible for a programmer to consider all the ways a computing innovation can be used.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	☑	□	□	□	□	□	☑	☑

IOC-1.B.5 Often computing innovations have had a beneficial effect by leading to advances in other fields.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	☑	□	□	□	□	□	□	☑

IOC-1.B.6 Rapid sharing of the program or the results of running a program with a large number of users can result in significant impacts beyond the intended purpose or control of the programmer.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	□	□	□	□	□	□	□	☑

IOC-1.C Describe issues that contribute to the digital divide.

IOC-1.C.1 Internet access varies between socioeconomic, geographic, or demographic characteristics and between countries.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	□	□	□	□	□	□	□	☑

IOC-1.C.2 The “digital divide” refers to differing access to computing devices and the Internet, based on socioeconomic, geographic, or demographic characteristics.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	□	□	□	□	□	□	☑	☑

IOC-1.C.3 The digital divide can affect both groups and individuals.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	□	□	□	□	□	□	☑	☑

IOC-1.C.4 The digital divide raises issues of equity, access, and influence, both globally and locally.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	□	□	□	□	□	□	☑	☑

IOC-1.C.5 The digital divide is affected by the actions of individuals, organizations, and governments.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	□	□	□	□	□	□	☑	☑

IOC-1.D Explain how bias exists in computing innovations.

IOC-1.D.1 Computing innovations can reflect existing human biases because of biases written into the algorithms or biases in the data used by the innovation.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	☑	□	□	□	□	□	□	☑

IOC-1.D.2 Programmers should take action to reduce bias in algorithms used for computing innovations as a way of combating existing human biases.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	☑	□	□	□	□	□	☑	☑

IOC-1.D.3 Biases can be embedded at all levels of software development.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	☑	□	□	□	□	□	☑	☑

IOC-1.E Explain how people participate in problem-solving processes at scale.

IOC-1.E.1 Widespread access to information and public data facilitates the identification of problems, development of solutions, and dissemination of results.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	☑	□	□	□	□	□	☑	☑

IOC-1.E.2 Science has been impacted by using scale and “citizen science” to solve scientific problems.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	☑	□	□	□	□	□	□	☑

IOC-1.E.3 Citizen science is scientific research conducted in whole or part by distributed individuals, many of whom may not be scientists, who contribute relevant data to research using their own computing devices.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	☑	□	□	□	□	□	□	☑

IOC-1.E.4 *Crowdsourcing* is the practice of obtaining input or information from a large number of people via the Internet.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	☑	□	□	□	□	□	□	☑

IOC-1.E.5 Human capabilities can be enhanced by collaboration via computing.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	☑	□	□	□	□	□	□	☑

IOC-1.E.6 Crowdsourcing offers new models for collaboration, such as connecting people with jobs and businesses or social causes with funding.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	☑	□	□	□	□	□	□	☑

IOC-1.F Explain how the use of computing can raise legal and ethical concerns.

IOC-1.F.1 Material created on a computer is the intellectual property of the creator or an organization.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	☑	□	☑	□	□	□	□	□

IOC-1.F.2 Ease of access and distribution of digitized information raises intellectual property concerns regarding ownership, value, and use.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	☑	□	☑	□	□	□	□	□

IOC-1.F.3 Measures should be taken to safeguard intellectual property.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	☑	☑	☑	□	□	□	□	□

IOC-1.F.4 The use of material created by someone else without permission and presented as one's own is plagiarism and may have legal consequences.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	☑	□	☑	□	□	□	□	□

IOC-1.F.5 Some examples of legal ways to use materials created by someone else include:

- Creative Commons - a public copyright license that enables the free distribution of an otherwise copyrighted work. This is used when the content creator wants to give others the right to share, use, and build upon the work they have created.
- open source – programs that are made freely available and may be redistributed and modified.
- open access - online research outputs that are free of any and all restrictions on access, and free of many restrictions on use such as copyright or license restrictions.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	☑	□	☑	□	□	□	□	□

IOC-1.F.6 The use of material created by someone other than yourself should always be cited.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	☑	☑	□	□	□	□	□	□	□

IOC-1.F.7 Creative commons, open source, and open access have enabled broad access to digital information.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☐	☑	☐	☑	☐	☐	☐	☐	☐

IOC-1.F.8 Using computing to harm individuals or groups of people raise legal and ethical concerns.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☑	☑	☐	☑	☐	☐	☐	☐	☐

IOC-1.F.9 Computing can play a role in social and political issues which in turn often raise legal and ethical concerns.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☑	☑	☐	☐	☐	☐	☐	☐	☐

IOC-1.F.10 The digital divide raises ethical concerns around computing.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☐	☑	☐	☐	☐	☐	☐	☐	☐

IOC-1.F.11 Computing innovations can raise legal and ethical concerns. Some examples of these include:

- the development of software that allows commercial access to music and movie downloads and streaming
- the development of algorithms that include bias
- the existence of computing devices that are always listening and recording what you are saying

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☐	☑	☐	☑	☐	☐	☐	☐	☑

IOC-2.A Describe the risks to privacy from collecting and storing personal data on a computer system.

IOC-2.A.1 Personally identifiable information (PII) is information about an individual that identifies, links, relates, or describes them. Examples of PII include:

- social security number
- age
- race
- phone number(s)
- medical information
- financial information
- biometric data

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☑	☐	☐	☐	☐	☐	☐	☐	☐

IOC-2.A.2 Search engines can record and maintain a history of searches made by users.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☑	☐	☐	☐	☐	☐	☐	☐	☐

IOC-2.A.3 Websites can record and maintain a history of individuals who have viewed their pages.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☐	☐	☐	☑	☐	☐	☐	☐	☐	☐	☐	☐

IOC-2.A.4 Devices, websites, and networks can collect information about a user’s location.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	☑	□	□	□	□	□	□	□

IOC-2.A.5 Technology enables the collection, use, and exploitation of information about, by, and for individuals, groups, and institutions.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	☑	□	□	□	□	□	□	□	□

IOC-2.A.6 Search engines can use search history to suggest websites or for target marketing.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	☑	□	□	□	□	□	□	□	□

IOC-2.A.7 Disparate personal data, such as geolocation, cookies, and browsing history, can be aggregated to create knowledge about an individual.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	☑	□	□	□	□	□	□	□	□

IOC-2.A.8 PII and other information placed online can be used to enhance a user’s online experiences.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	☑	□	□	□	□	□	□	□	□

IOC-2.A.9 PII stored online can be used to simplify making online purchases.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	☑	□	□	□	□	□	□	□	□

IOC-2.A.10 Commercial and governmental curation of information may be exploited if privacy and other protections are ignored.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	☑	□	□	□	□	□	□	□	□

IOC-2.A.11 Information placed online can be used in ways that were not intended and that may have a harmful impact. For example, an email message may be forwarded, tweets can be retweeted, and social media posts can be viewed by potential employers.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	☑	□	□	□	□	□	□	□	□

IOC-2.A.12 PII can be used to stalk or steal the identity of a person, or to aid in the planning of criminal acts.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	☑	□	□	□	□	□	□	□	□

IOC-2.A.13 Once information is placed online, it is difficult to delete.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	☑	□	□	□	□	□	□	□	□

IOC-2.A.14 Programs can collect your location and record where you have been, how you got there, and how long you were at a given location.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	☑	□	□	□	□	□	□	□	□

IOC-2.A.15 Information posted to social media services can be used by others. Combining information posted on social media and other sources can be used to deduce private information about you.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	☑	□	□	□	□	□	□	□	□

IOC-2.B Explain how computing resources can be protected and can be misused.

IOC-2.B.1 Authentication measures protect devices and information from unauthorized access. Examples of authentication measures include strong passwords and multifactor authentication.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	☑	□	□	□	□	□	□	□	□

IOC-2.B.2 A strong password is something that is easy for a user to remember but would be difficult for someone else to guess based on knowledge of that user.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	☑	□	□	□	□	□	□	□	□

IOC-2.B.3 *Multifactor authentication* is a method of computer access control in which a user is only granted access after successfully presenting several separate pieces of evidence to an authentication mechanism, typically in at least two of the following categories: knowledge (something they know); possession (something they have), and inherence (something they are).

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	☑	□	□	□	□	□	□	□	□

IOC-2.B.4 Multifactor authentication requires at least two steps to unlock protected information; each step adds a new layer of security that must be broken to gain unauthorized access.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	☑	□	□	□	□	□	□	□	□

IOC-2.B.5 *Encryption* is the process of encoding data to prevent unauthorized access. *Decryption* is the process of decoding the data. Two common encryption approaches are:

- Symmetric key encryption involves one key for both encryption and decryption.
- Public key encryption pairs a public key for encryption and a private key for decryption. The sender does not need the receiver’s private key to encrypt a message, but the receiver’s private key is required to decrypt the message.

Exclusion Statement (EK IOC-2.B.5):
Specific mathematical procedures for encryption and decryption are beyond the scope of this course and the AP Exam.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	☑	□	☑	□	□	□	□	□	□

IOC-2.B.6 Certificate authorities issue digital certificates that validate the ownership of encryption keys used in secure communications and are based on a trust model.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	☑	□	□	□	□	□	□	☑	□

IOC-2.B.7 Computer virus and malware scanning software can help protect a computing system against infection.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	☑	□	□	□	□	□	□	□	□

IOC-2.B.8 A *computer virus* is a malicious program that can copy itself and gain access to a computer in an unauthorized way. Computer viruses often attach themselves to legitimate programs and start running independently on a computer.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	☑	□	□	□	□	□	□	☑	□

IOC-2.B.9 *Malware* is software intended to damage a computing system or to take partial control over its operation.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	☑	□	□	□	□	□	□	□	□

IOC-2.B.10 All real-world systems have errors or design flaws that can be exploited to compromise them. Regular software updates help to fix errors that comprise a computing system.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	☑	□	□	□	□	□	□	□	□

IOC-2.B.11 Users can control the permissions applications have for collecting user information. Users should review the permissions of applications to protect their privacy.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	☑	□	□	□	□	□	□	□	□

IOC-2.C Explain how unauthorized access to computing resources is gained.

IOC-2.C.1 *Phishing* is a technique that attempts to trick a user into providing personal information. That personal information can then be used to access sensitive online resources, such as bank accounts and emails.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	☑	□	□	□	□	□	□	□	□

IOC-2.C.2 *Keylogging* is the use of a program to record every keystroke made by a computer user in order to gain fraudulent access to passwords and other confidential information.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	☑	□	□	☑	□	□	□	□	□

IOC-2.C.3 Data sent over public networks can be intercepted, analyzed and modified. One way that this can happen is through a rogue access point.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	☑	☑	□	□	□	□	□	□	☑

IOC-2.C.4 A *rogue access point* is a wireless access point that gives unauthorized access to secure networks.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	☑	☑	□	□	□	□	□	□	☑

IOC-2.C.5 A malicious link can be disguised on a web page or in an email message.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	☑	□	□	□	□	□	□	□	□

IOC-2.C.6 Unsolicited emails, attachments, links, and forms in emails can be used to compromise the security of a computing system. These can come from unknown senders or from known senders whose security has been compromised.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	☑	□	□	□	□	□	□	□	□

IOC-2.C.7 Untrustworthy (often free) downloads from freeware or shareware sites can contain malware.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	☑	□	□	□	□	□	□	□	□

D12 Computational Thinking Practices in CSP

The transferable practices and skills students will develop that are fundamental to the discipline of computer science.

1 Computational Solution Design - Design and evaluate computational solutions for a purpose.

1A Analyze the situation, context or task.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	□	□	□	□	□	☑	□	□

1B Determine and design an appropriate method or approach to achieve the purpose.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	☑	□	□	☑	□	□	☑	□	□	☑

1C Explain how collaboration affects the development of solutions.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	☑	☑	☑	☑	☑	☑	☑	□	☑	☑

1D Evaluate solution options.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	☑	☑	☑	☑	☑	□	☑	□	□	□

2 Algorithms and Program Development - Develop and implement algorithms.

2A Represent algorithmic processes without using a programming language.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	□	□	□	□	□	□	□	□	□	□	□

2B Implement an algorithm in a program.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	☑	☑	☑	☑	□	□	☑	□	□	□

3 Abstraction in Program Development - Develop programs that incorporate abstractions.

3A Generalize data sources through variables.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	□	☑	☑	☑	☑	□	□	☑	□	□

3B Use abstraction to manage complexity in a program.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	☑	☑	□	☑	☑	□	□	□	□	□	□

3C Explain how abstraction manages complexity.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	☑	☑	□	☑	☑	□	□	□	□	□	□

4 Code Analysis - Evaluate and test algorithms and programs.

4A Explain how a code segment of program functions.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	☑	☑	☑	☑	☑	☑	☑	□	□	□

4B Determine the result of code segments.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	□	□	□	□	□	□	□	□	□	□

4C Identify and correct errors in algorithms and programs.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	□	□	☑	□	□	□	□	□	□	□	□

5 Computing Innovations - Investigate computing innovations.

5A Explain how computing systems work.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	☑	□	□	□	□	□	□	□

5B Explain how knolwedge can be generated from data.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	□	□	☑	☑	☑	□	□	□

5C Describe the impact of a computing innovation.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	☑	□	□	□	□	□	☑	□

5D Describe the impact of gathering data.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	□	□	☑	☑	☑	□	□	□

5E Evaluate the uses of computing based on legal and ethical factors.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	☑	□	☑	☑	□	☑	□	☑	☑	□	□

6 Responsible Computing - Contribute to an inclusive, safe, collaborative, and ethical computing culture.

6A Collaboration in the development of solutions.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑

6B Use safe and secure methods when using computing devices.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	☑	□	□	□	□	□	□	□	□

6C Acknowledge the intellectual property of others.

Lesson	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
	□	□	□	□	☑	□	☑	□	□	□	□	□