Outline of BJC Unit 1: Introduction to Snap! Programming

Description

Programming

In the very first lab, students dive right into creating a simple game-app that they can share with their smart-phone. They learn several technical details: how to log in; save work; build scripts; "package" a script by building a custom block (procedure) with input parameters; get a sprite to draw, follow a motion, and interact with another sprite. Then they begin their second creative project, building a block that can produce grids of various dimensions for games like Tic Tac Toe and Sudoku. They learn how Snap*!* can perform calculations for them, and how to use those calculations to automate the process of fitting the game board grids to the screen. The projects emphasize the sensible use of abstraction as an organizing and simplifying tool.

Social Implications

Students discuss social implications of computing, including video games, privacy, and an introduction to the general question of whether or to what extent technology is good or bad.

Big Ideas 1, 2, 4, 5, 7

Students are introduced to foundational concepts of **programming (Big Idea 5)** including loops, variables, and procedures, and apply these in small programming projects in which they are encouraged to explore creatively and embellish their programs to support the **creative (Big Idea 1)** development of interesting computational artifacts. As students learn to develop and evaluate **algorithms (Big Idea 4)** for implementation and execution on a computer, they see that complex programs can be simplified by using **abstraction (Big Idea 2)** to manage the complexity. Unit 1 includes a Social Implications lab, beginning a study of **global impact (Big Idea 7)** that is maintained throughout the course. The Unit 1 Social Implications topic is on privacy, violent video games, and innovations.

Computational Thinking P3, P4, P5, P6

One way students develop their computational thinking skills is through programming and analyzing the programs of others. For example, students are asked to predict what they think will result from a given Snap*!* script. Making such predictions requires students to make sense of the sequential nature of the code. Students are introduced to one use of **abstracting (P3)** in packaging a collection of details in a form that more clearly expresses their meaning or purpose. They will expand their understanding of abstraction throughout the course. The emphasis on Pair Programming in Unit 1 establishes the regular practice of **communicating (P5)** and **collaborating**

(P6) to solve problems. Students **analyze problems (P4)** about the broader world of computing by exploring its impacts. Students cultivate analytic thinking skills as they debate and sometimes write about important and complex social issues.

Enduring Understandings

- **EU 2.2** Multiple levels of abstraction are used to write programs or create other computational artifacts.
- **EU 3.3** There are trade-offs when representing information as digital data.
- **EU 4.1** Algorithms are precise sequences of instructions for processes that can be executed by a computer and are implemented using programming languages.
- **EU 5.1** Programs can be developed for creative expression, to satisfy personal curiosity, to create new knowledge, or to solve problems (to help people, organizations, or society).
- **EU 5.2** People write programs to execute algorithms.
- **EU 5.3** Programming is facilitated by appropriate abstractions.
- **EU 5.5** Programming uses mathematical and logical concepts.
- **EU 7.3** Computing has a global effect — both beneficial and harmful — on people and society.
- **EU 7.4** Computing innovations influence and are influenced by the economic, social, and cultural contexts in which they are designed and used.

Learning Objectives

- **LO 2.2.1** Develop an abstraction when writing a program or creating other computational artifacts. [P2]
- **LO 3.3.1** Analyze how data representation, storage, security, and transmission of data involve computational manipulation of information. [P4]
- **LO 4.1.1** Develop an algorithm for implementation in a program. [P2]
- **LO 4.1.2** Express an algorithm in a language. [P5]
- **LO 5.1.2** Develop a correct program to solve problems. [P2]
- **LO 5.1.3** Collaborate to develop a program. [P6]

- **LO 5.2.1** Explain how programs implement algorithms. [P3]
- **LO 5.3.1** Use abstraction to manage complexity in programs. [P3]
- **LO 5.4.1** Evaluate the correctness of a program. [P4]
- **LO 5.5.1** Employ appropriate mathematical and logical concepts in programming. [P1]
- **LO 7.3.1** Analyze the beneficial and harmful effects of computing. [P4]
- **LO 7.4.1** Explain the connections between computing and economic, social, and cultural contexts. [P1]

Major Activities/Requirements for Completing Unit

Activities and Learning Opportunities	**Relationship to the Enduring Understandings**	**Relationship to the Learning Objectives**
Lab 1: Building an App • Students build a simple game they can play on their phones, in which players try to click on a character as it moves around the screen.	Students write programs to execute algorithms (**EU 5.2**) that implement a sequence of instructions for the computer to follow (**EU 4.1**). This lab builds toward these EUs by focusing on a simple game algorithm in which students explore procedures that control sprite appearance and movement. This lab provides line-by-line support for project development as students familiarize themselves with the Snap*!* programming interface.	Students collaborate to develop a program in Snap*!* (**LO 4.1.2**) to execute algorithms that control sprite appearance and movement (**LO 5.1.3**) in order to implement the desired behavior in a simple game (**LO 4.1.1**).
Lab 2: Sprite Drawing and Interaction • Students experiment with basic sprite commands, including `turn` and `move`, and use loops to create scripts in which sprites draw and	Students continue to write programs to execute algorithms (**EU 5.2**), ordered sequences of instructions (**EU 4.1**), throughout the curriculum. Now, students combine commands to	Students analyze how several programs implement algorithms (**LO 5.2.1**), e.g. discussing how a short algorithm with a loop generates the image of a square; comparing and contrasting various square-drawing algorithms; and predicting what

follow each other or the mouse.	develop programs according to their own ideas, curiosity, and quest for knowledge (**EU 5.1**). This lab builds toward these EUs by focusing on slightly more advanced algorithms involving loops and sprite orientation, positioning, and communication. This lab includes mini-projects that allow students to make creative choices including messages to the user and drawing experiments.	an algorithm does.
Lab 3: Building Your Own Blocks • Students create a "draw square" procedure that accepts a "size" variable input and design and debug other procedures that call the first procedure several times to create a design.	Students use the abstraction of a "draw square" algorithm to facilitate (**EU 5.3**) the writing of programs with a higher level of abstraction (**EU 2.2**) that call the "draw square" algorithm. This lab builds toward these EUs by focusing on custom procedures. The lab requires students to create several short scripts (some of which accept input values) and to use scripts within other scripts.	Students develop the abstraction (**LO 2.2.1**) of a "draw square" procedure to manage complexity (**LO 5.3.1**) in a program that draws designs with multiple squares. They also learn to debug their programs (**LO 5.4.1**) and how to use debugging practices to develop bug-free programs (**LO 5.1.2**).
Lab 4: Building Grids for Games • Students differentiate between *commands*, which carry out actions, and *reporters*, which report a value and can be used as inputs.	Students learn about mathematical and logical concepts (**EU 5.5**) as they evaluate expressions needed to draw and position a game board on the stage. This lab builds toward this EU by focusing on several mathematical ideas essential to computer science including formalizing screen positioning and orientation, syntax	Students employ the following mathematical and logical concepts of programming (**LO 5.5.1**): rounding, modulo, numerical operators (e.g., `+`), and Boolean values (`True`, `False`), and analyze code with nested operations.

	of order of operations, rounding, modulo, and Boolean values and operators. The lab requires students to evaluate arithmetic expressions without the Snap*!* interface and to experiment with less-familiar operations.	
Lab 5: Explosion of Bits; Games and Violence • Students read about and discuss ways the world has been changed by computing, consider their own experiences with technology, compile a list of computing innovations, and seek evidence for ways that technology can be both good and bad.	Through readings, discussion, and debate, students consider some of the trade-offs of representing information digitally (**EU 3.3**); how computing offers both benefits and risks to people and society (**EU 7.3**); and the relationships between computing innovations and the contexts in which they are used (**EU 7.4**). This lab builds toward these EUs by focusing on impacts of technology on students' lives, technological innovations, issues of privacy, and violence in video games. The lab requires students to write a technology autobiography, brainstorm computing innovations, read about some of the impacts of technology on society, and discuss and debate impacts of technology on human lives.	Students analyze the effects of computing as they debate the impact of violent video games on youth violence (**LO 7.3.1**); consider the implications of data practices (e.g. privacy, free-speech, etc.) involved in managing and manipulating people's data (**LO 3.3.1**); and connect computing to social and cultural contexts (**LO 7.4.1**).

Outline of BJC Unit 2: Conditionals and Abstraction

Description

Programming

The programming labs of Unit 2 introduce conditionals, predicates, and script variables, and use small projects in language/linguistics, music, art, sprite-interaction, simple encryption, and mathematics both as contexts for practice of the computer science concepts and also to illustrate the broad variety of directions students' own projects can take. Lab 1 also introduces the power of lists, foreshadowing a deeper study of lists in Unit 3.

Unit 2 programming labs continue a focus on structure and abstraction. Starting with the challenging task of teaching the computer to generate the plurals of nouns (e.g., butterfly → butterflies, moth → moths, bush → bushes), students begin thinking about the *structure of programs*. They learn to test for special cases, and how to use, sequence, and optimize the conditionals that control a program based on those cases. They learn to chunk details of a process into meaningful, recognizable parts, creating "specialist" blocks so that the main top-level block can show the structure of the program un-camouflaged by the details. This makes reading and debugging code easer, and allows the "specialist" blocks to be refined without requiring revision to the overall program. All of these are at the heart of the Abstraction standard of AP CS Principle and are core to mathematical thinking as well as computer science.

Students also begin to think about what makes correctly working code "good" code—is it the brevity, the clarity, or some combination? (They do not yet encounter situations in which the speed-efficiency of the code can be a criterion.) This, too, helps them begin to attend to structure. They learn to think about debugging by deliberately looking for ways to make a program fail, and then finding ways to avert failures.

Social Implications

Students consider the innovations around us that collect data about us, the availability of information online, why privacy is good to protect, reasons for giving up privacy and how to best they can protect their own online privacy. Students also examine communications technology and ways that computing impacts community (including cyberbullying).

Big Ideas 1, 2, 4, 5, 7

The major **programming (Big Idea 5)** focus of Unit 2 is on structure and **abstraction (Big Idea 2)**. Students learn to chunk details of **algorithms (Big Idea 4)** into meaningful, recognizable parts by creating "specialist" procedures so that the top-level procedure can show the structure of the program un-camouflaged by the details. This makes reading and debugging code easer and allows the "specialist" procedures to be reused in other algorithms and to be refined without requiring revision to higher-level procedures.

Students are offered a choice of programming projects designed to strengthen these ideas with **creative (Big Idea 1)** tasks. The unit also includes a Social Implications lab on data privacy, cyberbullying, and impacts on community building, addressing **global impact (Big Idea 7).**

Computational Thinking P1, P2, P3, P4, P5, P6

The increased challenge of Unit 2's programs gives pair programmers good reason and plenty of practice to develop **communication (P5)** and **collaboration (P6)** skills because those skills are genuinely needed as students create these more complex **computational artifacts (P2)**. Over the course of the year, the discussions of social implications also help to develop those communication and collaboration skills and create a context for broader discussion of hard social issues.

The focus on structure, **abstraction (P3)**, and debugging in Unit 2 also supports students **analyzing problems and digital artifacts (P4)** for they must determine where abstraction is needed and how to implement it and then critique their own code in their attempt to improve it and resolve errors.

In the Social Implications Labs, students **connect computing (P1)** to issues they can identify with, both the impacts of computing and the connections between people and computing.

Enduring Understandings

- **EU 1.1** Creative development can be an essential process for creating computational artifacts.
- **EU 1.2** Computing enables people to use creative development processes to create computational artifacts for creative expression or to solve a problem.
- **EU 1.3** Computing can extend traditional forms of human expression and experience.
- **EU 2.2** Multiple levels of abstraction are used to write programs or create other computational artifacts.
- **EU 3.2** Computing facilitates exploration and the discovery of connections in information.
- **EU 3.3** There are trade-offs when representing information as digital data.
- **EU 4.1** Algorithms are precise sequences of instructions for processes that can be executed by a computer and are implemented using programming languages.
- **EU 5.1** Programs can be developed for creative expression, to satisfy personal curiosity, to create new knowledge, or to solve problems (to help people, organizations, or society).

- **EU 5.2** People write programs to execute algorithms.
- **EU 5.3** Programming is facilitated by appropriate abstractions.
- **EU 5.4** Programs are developed, maintained, and used by people for different purposes.
- **EU 5.5** Programming uses mathematical and logical concepts.
- **EU 7.1** Computing enhances communication, interaction, and cognition.
- **EU 7.3** Computing has a global effect — both beneficial and harmful — on people and society.

Learning Objectives

- **LO 1.1.1** Apply a creative development process when creating computational artifacts. [P2]
- **LO 1.2.1** Create a computational artifact for creative expression. [P2]
- **LO 1.2.4** Collaborate in the creation of computational artifacts. [P6]
- **LO 1.3.1** Use computing tools and techniques for creative expression. [P2]
- **LO 2.2.1** Develop an abstraction when writing a program or creating other computational artifacts. [P2]
- **LO 2.2.2** Use multiple levels of abstraction to write programs. [P3]
- **LO 3.2.2** Use large data sets to explore and discover information and knowledge. [P3]
- **LO 3.3.1** Analyze how data representation, storage, security, and transmission of data involve computational manipulation of information. [P4]
- **LO 4.1.1** Develop an algorithm for implementation in a program. [P2]
- **LO 5.1.2** Develop a correct program to solve problems. [P2]
- **LO 5.1.3** Collaborate to develop a program. [P6]
- **LO 5.2.1** Explain how programs implement algorithms. [P3]
- **LO 5.3.1** Use abstraction to manage complexity in programs. [P3]
- **LO 5.4.1** Evaluate the correctness of a program. [P4]

- **LO 5.5.1** Employ appropriate mathematical and logical concepts in programming. [P1]
- **LO 7.1.1** Explain how computing innovations affect communication, interaction, and cognition. [P4]
- **LO 7.1.2** Explain how people participate in a problem-solving process that scales. [P4]
- **LO 7.3.1** Analyze the beneficial and harmful effects of computing. [P4]

Major Activities/Requirements for Completing Unit

Activities and Learning Opportunities	**Relationship to the Enduring Understandings**	**Relationship to the Learning Objectives**
Lab 1: Conditional Blocks • Students use conditionals (especially `if-else` and `if`) and predicates (such as < or =) to control the behavior of their programs. Using multiple `if` statements to distinguish multiple conditions, students design one program that sets a traffic signal to the inputted color and another to decide how to make the plural of a noun.	Throughout Unit 2, students continue to design sequences of instructions to solve problems (**EU 4.1**) and to write programs to execute these algorithms (**EU 5.2**). In order to create a program that correctly constructs the plural form of an input word (**EU 5.1**), students use abstraction—in this case, specialist procedures—to manage complexity (**EU 5.3**). This lab builds toward these EUs by focusing on conditionals (`if-else` and `if`) to control the behavior of a program. The lab requires students to develop a program that uses predicates to test for various conditions and use conditional blocks to control programs based on those conditions. This lab also focuses on abstraction through analyzing tasks to break them into subtasks and then creating blocks that specialize in these subtasks and	As they do throughout the unit, students collaborate (**LO 5.1.3**) as they develop an algorithm (**LO 4.1.1**). Students develop specialist procedures to handle special cases of singular nouns in specific ways, (**LO 2.2.1**) to manage the complexity, readability, and debuggability of their plurals program (**LO 5.3.1**). Students analyze a script to determine what is likely to go wrong before running it (**LO 5.4.1**) and use this developing understanding to create correct programs to calculate powers of numbers and count the number of vowels in a sentence (**LO 5.1.2**).

	through analyzing and debugging scripts.	
Lab 2: Script Variables • Building on their experience with input variables, students build local variables (called "script variables" in Snap!), and choose from among a set of small projects that offer practice with building selection procedures and debugging.	As students choose from among a variety of programs to develop, they have the opportunity to appreciate the range of purposes for which people create programs (**EU 5.4**), whether for creative expression, personal curiosity, knowledge generation, or problem solving (**EU 5.1**) . Students develop their computational artifacts for creative expression or to solve a problem (depending on the project chosen) (**EU 1.2**) and experience how computing can extend traditional forms of human expression and experience (**EU 1.3**). This lab builds toward these EUs by focusing on the `script variables` procedure, which allows students to create local variables.	Students collaborate (**LO 1.2.4**) to create a program—a computational artifact—for creative expression in art, language, or mathematics (**LO 1.2.1**) by using Snap*!* and the graphics and information-processing techniques they have learned (**LO 1.3.1)**. As a class, students predict how algorithms will be implemented (**LO 5.2.1**).
Lab 3: Tools and Techniques • Students work through four short activities—essentially how-to's—showing techniques that are useful in more complex programming tasks. Students get a bit more experience with Boolean expressions. They learn how to indicate the intended *type* of an input to a block—whether the intended input is to be a number, a	Students work on programs using mathematical and logical concepts (**EU 5.5**). This lab builds toward these EUs by focusing on the mathematics of computer science including basic arithmetic operations, the round block, the mod block, the random block, and logical operators. The lab also provides practice composing functions.	Students learn about object type—e.g., text, number, string—and learn how to specify that an input is intended to be a number. They focus attention on number as one of the fundamental object types in programming (**LO 5.5.1**).

string, a list, or something else. They get a bit more practice creating reporter blocks that take multiple inputs, and they use a composition of functions on those inputs. And they compare multiple *correct* ways of solving a problem, thinking about the esthetics of programming.		
Lab 4: Abstraction • Students build two complex games (a board game and a number guessing game) that focus on using abstraction to write clear, debuggable, and improvable code. Students are given goals, specifications, and guidance on structuring the programs, but they must find their own ways to build the parts. To start, students ease into abstraction by drawing a brick wall row-by-row and brick-by-brick.	Students continue to work on programs using multiple levels of abstraction (**EU 2.2**, **EU 5.3**). The challenges of these more complex tasks—particularly the challenges in developing the game board—require an iterative and exploratory process in order to achieve a properly working program (**EU 1.1**). This lab builds toward these EUs by focusing on ideas of abstraction to write clear, debuggable, improvable code. This lab provides goals and specifications for the programs and guidance on *structuring* them. The lab emphasizes abstraction through the development of procedures with a special purpose, and students must specify the type of mathematical and logical input that a procedure is expecting to receive.	Students use an iterative and exploratory development process (**LO 1.1.1**) and multiple levels of abstraction as they create these apps. Students use multiple-levels of abstraction as they draw a brick wall by calling a procedure that draws rows of bricks which in turn calls a procedure that draws an individual brick (**LO 2.2.1**, **LO 2.2.2**, **LO 5.3.1**).
Lab 5: Privacy; Community and Online	Building on their understanding of the	Students continue to analyze the impact of

Interactions • Students read about and/or discuss data-collecting innovations, privacy issues, ways in which computing affects our ability to build community, and cyberbullying (causes, impacts, and the roles of computing and social decisions about technology).	trade-offs of digital information (**EU 3.3**) and the potential benefits and harms of computing on society (**EU 7.3**), students more knowledgeably consider their own personal data and technology use. In researching and reflecting on the availability of their own personal information on the Internet, students discuss how computing allows people and organizations to know more about other people by making connections among data (**EU 3.2**). Students also discuss ways we use technology to communicate and interact in society (**EU 7.1**). This lab builds toward these EUs by asking students to focus on information that is available online about them (or someone they know), discuss why privacy is good to protect, and consider reasons for giving up that privacy. This lab provides students an opportunity to learn about innovations that collect data about people. The lab requires students to examine communications technology, discuss ways in which computing affects the ability to build community, and the positive and negative effects of certain computing technology, including social media and its capabilities.	computing (**LO 7.3.1**) as they consider how data management involves computational manipulation (**LO 3.3.1**). Reflecting on their own data and technology, students extend their insights to consider how large data sets of many people's data (**LO 3.2.2**) affect large-scale communication, interaction, collaboration, and problem-solving (**LO 7.1.1**, **LO 7.1.2**).

Outline of BJC Unit 3: Lists

Description

Programming

This unit focuses on lists, an *aggregate* data type for storing multiple items of any type, including numbers, strings, other lists, or even blocks. Just as functions can take numbers and strings as inputs, they can also take lists as input, or produce lists as output. A list is an ordered, numbered sequence of items. Similar data types in other programming languages may be called "arrays," "sequences," or "vectors."

This unit also focuses on several powerful list-processing functions, including the **higher-order functions** `map`, `keep`, and `combine`. These functions are called "higher order" because, along with other data, they take functions as inputs. For example, often a programmer wants to compute some function of each item in a list. Instead of writing separate procedures such as "take the first letter of each item," "add 3 to each item," etc., the `map` function generalizes the "... of each item" part, and takes another function as input to specify the "first letter of" or "add 3 to" part.

One of several design features that distinguishes BJC from other CS Principles curricula is that we emphasize *functional* programming. One virtue of the higher order list functions is that they generate new lists to report, rather than mutate existing lists. This is in contrast with the imperative, looping, mutation-based programming that is more common, but more error-prone, in dealing with sequential data. No attention (for now) needs to be placed on the idea of procedures as data; the Snap*!* visual representation of higher order functions makes the use of a function (rather than the value it reports) as input apparent at a glance. The grey ring that represents a procedure as data is already in the higher order function block, and the user of the higher order function doesn't have to do anything else to make the function itself, rather than a value it reports, be taken as the input. Near the end of the course, students build these higher order functions for themselves.

In situations in which imperative programming is needed, we still use a higher order procedure, `for each item`, a C-shaped block that takes a list and a script to be run for each item in the list. This avoids the need for index variables.

Social Implications

The social implications topics in Unit 3 are *search* and *encryption (part 1)*. In both cases, the readings from *Blown to Bits* describe the underlying technology and also raise questions about risks. The BJC lab pages focus more on the latter. In the case of searching, the main issues are about user profiling (for serving advertising and for tailoring search results) and about bias in results (profile-based or

otherwise). For encryption, we do emphasize how revolutionary public key cryptography has been, but the main emphasis is on how the various social stakeholders view the question of strong encryption of user data.

Big Ideas 2, 3, 4, 5, 7

As students work with lists to manage **data and information (Big Idea 3)**, they use **abstraction (Big Idea 2)**, such as abstract data types, to manage the complexity of datasets. They also create **algorithms (Big Idea 4)** using higher-order functions and implement these algorithms in various **programs (Big Idea 5)** that use lists and list functions.

In the Social Implications Labs, students continue their study of the **global impact (Big Idea 7)** of computing, now by focusing on the impact and storage of user data.

Computational Thinking P1, P2, P3, P6

Lists are an important data structure, and learning to think of lists as a single object (rather than many objects) is a way students practice **abstracting (P3)**. We introduce the idea of implementing an abstract data type by writing constructor and selector functions. Students continue to **collaborate (P6)** through pair programming.

Each lab is built around a small "app" project, such as a contact list app, a **computational artifact (P2)** created by the student. This **connects computing (P1)** with the cell phone apps students understand.

Enduring Understandings

- **EU 2.1** A variety of abstractions built on binary sequences can be used to represent all digital data.
- **EU 2.2** Multiple levels of abstraction are used to write programs or create other computational artifacts.
- **EU 3.2** Computing facilitates exploration and the discovery of connections in information.
- **EU 5.1** Programs can be developed for creative expression, to satisfy personal curiosity, to create new knowledge, or to solve problems (to help people, organizations, or society).
- **EU 5.2** People write programs to execute algorithms.
- **EU 5.3** Programming is facilitated by appropriate abstractions.
- **EU 5.5** Programming uses mathematical and logical concepts.
- **EU 6.3** Cybersecurity is an important concern for the Internet and the systems built on it.

- **EU 7.2** Computing enables innovation in nearly every field.
- **EU 7.5** An investigative process is aided by effective organization and selection of resources. Appropriate technologies and tools facilitate the accessing of information and enable the ability to evaluate the credibility of sources.

Learning Objectives

- **LO 1.2.2** Create a computational artifact using computing tools and techniques to solve a problem. [P2]
- **LO 1.2.5** Analyze the correctness, usability, functionality, and suitability of computational artifacts. [P4]
- **LO 2.2.1** Develop an abstraction when writing a program or creating other computational artifacts. [P2]
- **LO 2.2.2** Use multiple levels of abstraction to write programs. [P3]
- **LO 2.2.3** Identify multiple levels of abstractions that are used when writing programs. [P3]
- **LO 5.1.3** Collaborate to develop a program. [P6]
- **LO 5.3.1** Use abstraction to manage complexity in programs. [P3]
- **LO 5.5.1** Employ appropriate mathematical and logical concepts in programming. [P1]
- **LO 6.3.1** Identify existing cybersecurity concerns and potential options to address these issues with the Internet and the systems built on it. [P1]
- **LO 7.5.2** Evaluate online and print sources for appropriateness and credibility [P5]

Major Activities/Requirements for Completing Unit

Activities and Learning Opportunities	**Relationship to the Enduring Understandings**	**Relationship to the Learning Objectives**
Lab 1: Introduction to Lists • Students build a basic shopping list app and learn that lists can store data and that programs can access and manipulate list contents. They	Students develop algorithms (**EU 5.2**) to manage the storage and retrieval of data in a shopping list, lists of countries, and lists of words used to create comprehensible sentences, and they write programs that express and execute those algorithms (**EU**	Students collaborate (**LO 5.1.3**) to create a computational artifact using lists and list-processing procedures to organize information (**LO 1.2.2**).

also explore various list blocks by predicting and testing the outcome of given expressions and scripts. They have a first encounter with data abstraction and structure by using lists of words, classified by the function they perform in sentences, to build more complex structures, combining words into phrases and then combining these phrases into sentences.	**5.1**). This lab builds toward these EUs by introducing lists as a means to manage data for a variety of purposes, such as a practical shopping list app. The lab requires students to develop, program, and run algorithms to interact with the shopping list app's user and to generate English sentences based on lists of words organized by parts of speech.	
Lab 2: Nesting Lists • Students create a contact list app whose elements are themselves lists. Students learn how to cull a list to keep only the information they need, use loops to process lists of lists, and use data types to make lists of coordinate pairs.	Students use multiple levels of abstraction such as custom procedures that call custom procedures and abstract data types (**EU 2.2**). This lab builds toward this EU primarily by introducing data abstraction, using lists to represent abstract data types, including in one example an explicit type tag as part of the representation. The lab requires students to build a contacts app and to modify the code with and without data abstraction so that students experience directly how data abstraction makes programs more maintainable.	Students work with data abstraction at multiple levels, creating, using and modifying abstract data types. For example, they create a `get new contact` procedure that calls both a `get name` and a `get phone` procedure; they also work with a list of items whose elements are pairs of coordinates stored as an abstract data type `point` that students create (**LO 2.2.1, LO 2.2.2**, **LO 2.2.3**).

Lab 3: Three Key List Operations • Students use higher order functions (functions that take other functions as input) to process lists in various contexts.	Students use higher order functions (**EU 5.3**) to perform a variety of mathematical operations (**EU 5.5**). This lab builds toward these EUs by introducing higher order functions as a form of abstraction over algorithms and by using explicitly represented mathematical functions as inputs to higher order functions. In particular, this lab requires students to build predicate functions (for example, $x \rightarrow (x \bmod 2) = 1$ to select odd numbers from a list). Students are also required to explore, with examples, why non-associative functions aren't useful with the `combine` higher-order function.	Students use `map` (which applies an input function to each item of a list) to transpose music, perform geometric transformations on shapes, perform algebraic functions on list items, etc. They use `keep` (which filters a list) to select desired list items. And they use `combine` (which combines list items using an input binary operation) to sum, average, or find the maximum value in a list of numbers. These higher order functions are themselves abstractions (**LO 5.3.1**, **LO 5.5.1**).
Lab 4: Combining List Operations • Students practice using higher-order list-processing functions in several contexts both to build their facility and to learn to think in terms of what these functions can do. This lab does not introduce new ideas, but focuses on mastery and application.	Students think simultaneously about lists as a *single object* and, at the same time, as a set of *elements* to be processed (**EU 2.1**) as they build encoding/decoding functions and a tic-tac-toe game. This lab builds toward this EU with a mini-project in which students are required to use lists to represent text to be encyphered or decyphered and the moves in a tic-tac-toe game. Students must also use procedural abstraction; for example, a procedure to test whether player X has won in tic-tac-toe is built out of a procedure that reports a list of cells in which X has moved, a procedure that	Students compose higher order functions to develop a tic-tac-toe script that checks the board for wins (**LO 2.2.2**).

	reports all of the eight winning combinations of squares, and a function that tests whether a list of cells matches a particular winning combination.	
Lab 5: Search • Students learn about how search engines work, what is included and excluded from the results, the information that search engines can collect about your searches, and the implications of how the search engines might use that data. They also consider how *they* might design their own search engine, and in doing so, examine their own values and priorities.	Students consider what data search engines collect, how the data can be processed to discover connections in information (**EU 3.2**), and how innovations can come of access to information (**EU 7.2**). This lab builds toward these EUs by focusing attention on Internet search engines, which are an excellent example of how computers help with the discovery of connections within varied information and have contributed tremendously toward innovation. This lab requires students to compare results of different search engines and (primarily through readings from *Blown to Bits*) to explore the algorithms used by search engines. Students also consider the issues of privacy and result bias and how they might design a better search engine.	Students consider questions like: How important is privacy to me? What sources of information might be overlooked by the engineers who design these search engines at large companies? What phrases have special meaning to me that Yahoo or Google don't seem to know about? (**LO 1.2.5**).
Lab 6: Encryption • Students learn about encryption and decryption methods, consider why encryption is an issue, and engage in a debate in which they consider the viewpoints of	Students consider the value of cybersecurity to society (**EU 6.3**). They also choose and research an innovation that is affected by issues of encryption and do online research on their topic (**EU 7.5)**. This lab builds toward these EUs by	Students explore the basic programming concepts involved in cybersecurity **(LO 6.3.1**), and conduct online research requiring the evaluation of sources for appropriateness and credibility **(LO 7.5.2**).

governments, civil liberties groups, and businesses regarding the availability of encryption software.	considering both the technology (public key vs. private key) and the social implications of encryption. Students debate the issue of government access to encrypted data and explore how encryption supports non-obvious innovations, such as self-driving cars.	

Outline of BJC Unit 4: The Internet and Global Impact

Description

Programming

Unit 4 addresses the structure of the Internet, the various protocols on which it runs, and the implications of this technology to society. Students learn to recognize HTML and learn that it is another computer language, and they learn to "scrape" an HTML page for data. They build on their work in Unit 3 to analyze lists containing data using Snap*!* procedures and strategies they have already learned.

Social Implications

Students first consider what rules perhaps ought to exist regarding behavior or content on the Internet. They use the *Blown to Bits* reading and what they've learned about the Internet throughout this Unit to discuss the challenges of regulating the Internet, look at ways that countries (including the United States) approach regulation and censorship of online content, examine statistics about Internet usage around the world.

Students continue to familiarize themselves with the AP Explore Task requirements, specifically by examining an innovation that addresses a social issue to which they feel some personal connection. They first identify social issues they care about and learn some ways that technology has been used to address the issue and then use the AP Explore Task prompts to guide them towards thinking and writing about potential benefits and possible unintended negative consequences of the innovation.

Big Ideas 2, 3, 6

The purpose of this unit is to address the **Internet (Big Idea 6)** and begin to address **data and information (Big Idea 3)**, which is further addressed in Unit 5: Algorithms and Data. Students learn about the power of the fundamental **abstractions (Big Idea 2)** of the Internet including the two most essential protocols: IP, which builds one Internet out of a vast collection of local networks; and TCP, which allows the Internet to function reliably despite the lack of reliability of the physical infrastructure. Students also learn how the hierarchies of domain names and IP addresses make scalability possible.

Students also analyze and interpret a collection of GPS coordinates of potential clients for a (fictional) burger company by modeling the **data (Big Idea 3)**, identifying clusters of data, and finding the centers of those clusters (prospective restaurant locations).

Computational Thinking P1, P3, P4, P5, P6

Students learn to identify and describe the **abstractions (P3)** of the Internet, continue to **collaborate (P6)** as they pair program, and **communicate (P5)** as they **connect computing (P1)** to society in the Social Implications labs and describe the meaning of the results of their **analysis (P4)** of the GPS Data.

Enduring Understandings

- **EU 1.2** Computing enables people to use creative development processes to create computational artifacts for creative expression or to solve a problem.
- **EU 2.1** A variety of abstractions built on binary sequences can be used to represent all digital data.
- **EU 3.1** People use computer programs to process information to gain insight and knowledge.
- **EU 3.2** Computing facilitates exploration and the discovery of connections in information.
- **EU 4.1** Algorithms are precise sequences of instructions for processes that can be executed by a computer and are implemented using programming languages.
- **EU 5.4** Programs are developed, maintained, and used by people for different purposes.
- **EU 5.5** Programming uses mathematical and logical concepts.
- **EU 6.1** The Internet is a network of autonomous systems.
- **EU 6.2** Characteristics of the Internet influence the systems built on it.
- **EU 6.3** Cybersecurity is an important concern for the Internet and the systems built on it.
- **EU 7.1** Computing enhances communication, interaction, and cognition.
- **EU 7.3** Computing has a global affect — both beneficial and harmful — on people and society.
- **EU 7.4** Computing innovations influence and are influenced by the economic, social, and cultural contexts in which they are designed and used.
- **EU 7.5** An investigative process is aided by effective organization and selection of resources. Appropriate technologies and tools facilitate the accessing of information and enable the ability to evaluate the credibility of sources.

Learning Objectives

- **LO 1.2.2** Create a computational artifact using computing tools and techniques to solve a problem. [P2]
- **LO 1.2.3** Create a new computational artifact by combining or modifying existing artifacts. [P2]
- **LO 1.2.5** Analyze the correctness, usability, functionality, and suitability of computational artifacts. [P4]
- **LO 2.1.1** Describe the variety of abstractions used to represent data. [P3]
- **LO 2.1.2** Explain how binary sequences are used to represent digital data. [P5]
- **LO 3.1.1** Use computers to process information, find patterns, and test hypotheses about digitally processed information to gain insight and knowledge. [P4]
- **LO 3.1.2** Collaborate when processing information to gain insight and knowledge. [P6]
- **LO 3.1.3** Explain the insight and knowledge gained from digitally processed data by using appropriate visualizations, notations, and precise language. [P5]
- **LO 3.2.1** Extract information from data to discover and explain connections, patterns, or trends. [P1]
- **LO 4.1.1** Develop an algorithm for implementation in a program. [P2]
- **LO 4.1.2** Express an algorithm in a language. [P5]
- **LO 5.1.1** Develop a program for creative expression, to satisfy personal curiosity, or to create new knowledge. [P2]
- **LO 5.5.1** Employ appropriate mathematical and logical concepts in programming. [P1]
- **LO 6.1.1** Explain the abstractions in the Internet and how the Internet functions. [P3]
- **LO 6.2.1** Explain characteristics of the Internet and the systems built on it. [P5]
- **LO 6.2.2** Explain how the characteristics of the Internet influence the systems built on it. [P4]
- **LO 6.3.1** Identify existing cybersecurity concerns and potential options to address these issues with the Internet and the systems built on it. [P1]
- **LO 7.1.1** Explain how computing innovations affect communication, interaction, and cognition. [P4]
- **LO 7.3.1** Analyze the beneficial and harmful effects of computing. [P4]

- **LO 7.4.1** Explain the connections between computing and economic, social, and cultural contexts. [P1]
- **LO 7.5.1** Access, manage, and attribute information using effective strategies. [P1]
- **LO 7.5.2** Evaluate online and print sources for appropriateness and credibility [P5]

Major Activities/Requirements for Completing Unit

Activities and Learning Opportunities	**Relationship to the Enduring Understandings**	**Relationship to the Learning Objectives**
Lab 1: Website Data • After reading a brief primer on the Internet, students import web data into Snap*!*, split the lines of HTML into items in a list, and perform string operations to access information.	To support importing and processing HTML, students gain very basic familiarity with the code for graphic design algorithms (**EU 4.1)**. They also begin to learn about how the characteristics of the Internet influence the systems built on it (**EU 6.2**). This lab builds toward these EUs by focusing on the basic structures of the Internet including some of its history, network redundancy, domain name hierarchy, HTTP, HTML, and accessing web data via HTTP in Snap*!*. The lab offers written and video information on these characteristics of the Internet and requires students to develop algorithms to import CSV (comma separated values) data and organize it into a list of lists and also to import HTML.	Students learn about HTML and CSS, two common special-purpose languages (**LO 4.1.2**); the difference between the World Wide Web and the Internet; URLs; the physical structure and history of the Internet (**LO 6.1.1**); network redundancy; and the domain name hierarchy (**LO 6.2.1**, **LO 6.2.2**).

Lab 2: GPS Data • In this lab, students use Snap*!* to model and process a batch of GPS coordinates representing the locations of potential clients for a (fictional) new restaurant chain in New York City. Building on the work on reading data from Lab 1, students create an appropriately-scaled model of the data and analyze it to determine the optimal location for the new business.	Students develop an algorithm (**EU 4.1**) to process information (**EU 3.1**) and discover connections in the information—clusters representing where new restaurants should be built (**EU 3.2**). This lab builds toward these EUs by focusing on a business scenario—analyzing GPS coordinates to determine the ideal neighborhood for a new restaurant. The lab offers basic information on GPS coordinates and strategies for visualizing that data, and requires students to build a program to scale and plot the data and use the results to draw conclusions about where the new restaurant should be.	Students collaborate (**LO 3.1.2**) as they develop an algorithm to process the GPS data (**LO 4.1.1**), find patterns to gain new knowledge (**LO 3.1.1**, **LO 3.1.3**, **LO 5.1.1**), and discover and explain patterns and trends (**LO 3.2.1**).
Lab 3: Number Representation • Students learn about binary and hexadecimal representations, binary sequences, and how computers represent integers and non-integers.	Students learn about the abstractions that are value representations (**EU 2.1**) and translate among binary, decimal, and hexadecimal notation (**EU 5.5**). This lab builds toward these EUs by focusing on ideas of binary data, sequences, and representation. The lab offers explanation of floating point, binary and hexadecimal representation, and the use of HEX in RGB color and IP addresses. The lab also requires students to explore factorial outputs both with and without bignums to convert between binary, decimal, and hexadecimal	Students learn about bits and bit width (**LO 2.1.1**) and representations of digital data (**LO 2.1.2**), and they translate among three common number bases (**LO 5.5.1**).

	representations.	
Lab 4: Network Protocols • Students learn about the abstractions that make the Internet (especially TCP/IP), and the history and impact of the organizations controlling the communication standards and addressing systems for the Internet. Students also identify the IP address of the computer they are using by scraping the results of a "my ip address" query to a Server engine.	Students consider how the characteristics (e.g., technical limitations and open standards) of the autonomous system that is the Internet influences the systems on the network (**EU 6.1**, **EU 6.2**). This lab builds toward these EUs by focusing on the TCP/IP protocol. The lab presents text and video on IP address hierarchy, IPv4 and IPv6, TCP, and open standards. The lab requires students to develop a script to determine the computer's IP address, to explore a simulation of TCP, to discuss the fundamental unreliability of the internet and how we address that, and to review and discuss the basic abstractions of the Internet.	Students learn about IP address hierarchy, IPv4 vs. IPv6, packets and packet switching, reliable data transmission, open standards, the Internet abstraction hierarchy and how these systems interrelate (**LO 6.1.1**, **LO 6.2.1**, **LO 6.2.2**).
Lab 5: Weather App • Students create a weather app building on their work scraping data from web pages in Lab 1. They scrape a weather website for information, importing HTML to Snap*!*; produce a multi-city weather report; and use IP address information to determine current local conditions.	Scraping an existing development effort (a weather site) for relevant data, students create a weather app that solves the age-old problem of needing to know the weather (**EU 1.2**, **EU 3.1**). This lab builds toward these EUs by focusing on the development of a weather app. The lab requires students to send data to and scrape data from a weather website and to develop a presentation mechanism for that data.	Students collaborate (**LO 3.1.2**) to create a new computational artifact using the `http` procedure (**LO 1.2.2**) together with the weather website (**LO 1.2.3**) and then selecting and communicating specific information to the user (**LO 3.1.3**).

Lab 6: Internet Security • Students read about various cybersecurity risks and the basic concepts of cryptography. They develop a simple encryption script.	Students read about cybersecurity issues (e.g., lack of DNS security, common attacks) (**EU 6.3**). This lab builds toward this EU by focusing entirely on common security attacks and the ways that cryptography, open standards, and certificate authorities help to protect against these threats. The lab offers video and text about cybersecurity issues and requires students to develop a simple encryption program.	Students learn about common security attacks (bug exploits, viruses, phishing, and DDoS attacks) and learn about security measures they can take (**LO 6.3.1**).
Lab 7: Censorship and Computing around the World • Students read from *Blown to Bits* Chapter 7, consider what rules (if any) ought to exist on the Internet, and learn some of the practical difficulties of regulating Internet content. Students then consider issues of free speech and how global Internet leads to issues when different countries have different standards for what should be censored. They examine statistics for Internet use around the world.	Students learn about some of interpersonal risks of using the Internet (**EU 7.1**, **EU 7.4**) and the tension between free speech and security against these risks (**EU 6.1**, **EU 7.3**). This lab builds toward these EUs by focusing on the use and ethics of the Internet and some of the practical difficulties of regulating Internet content. The lab offers assigned reading, discussion prompts, and classroom activities. The lab requires students to reflect on how the nature of the Internet impacts issues of censorship, access, regulation, safety, and how computers are used to create computational artifacts.	Students consider questions like: Should people be allowed to say and post anything they want on the Internet? Who should be held responsible for the harm that could result in a case involving "cyber-predators" on a site like MySpace? Why should people in the United States care about Internet rules in other countries? This is also an opportunity for students to think further about their own role as digital citizens (**LO 6.3.1**, **LO 7.1.1**, **LO 7.3.1**, **LO 7.4.1**).

Lab 8: Innovating for Social Change • This lab helps students prepare for the AP Explore task by exposing them to the range of prompts they will encounter. They consider how technology might address a social issue they care about by identifying personally relevant social issues and learning ways technology has been used to make a difference in those areas.	Students consider the purposes (**EU 5.4**, **EU 7.4**) and social benefits of computing (**EU 7.1**, **EU 7.3)** by selecting a positive impact to research (**EU 7.5**). This lab builds toward these EUs by focusing on how technology can and cannot solve various social issues and how the same technological innovation can have both harms and benefits. The lab offers various reflection questions and discussion prompts to help students investigate these ideas deeply. The lab requires students to research and write about an innovation in preparation for the AP Explore Task.	Students research (**LO 7.5.1**, **LO 7.5.2**) and analyze the contextual suitability (**LO 1.2.5**, **LO 7.4.1**) and effects (**LO 7.1.1**, **LO 7.3.1**) of a computing innovation.

AP Through-Course Assessment: Explore – Impact of Computing Innovations Performance Task

- After completing Unit 4, students complete through-course assessment *Explore* - *Impact of Computing Innovations* (8 hours in class). The work in Unit 4 Lab 8 is prepares students well for completing this performance task.

Outline of BJC Unit 5: Algorithms and Data

Description

Programming

This unit focuses on several types of analysis: analysis of problems to generate algorithms for their solution; analysis of the algorithms (especially of the time it takes to execute them) in order to optimize them; analysis of phenomena by generating models and simulations that give insight and help one generate and test hypotheses; and analysis of data, especially including visualization.

Students have been generating algorithms to solve problems from the start of this course, but have not yet focused on analyzing them for efficiency. For small enough computational problems, such analysis isn't needed. But modeling complex phenomena and handling large data sets requires understanding that there are sometimes alternative algorithms that reduce the impact of the size of a model on the time it takes to execute. In-depth coverage of this broad domain (computational complexity, data analysis, modeling and simulation) is beyond the scope of an introductory course, but Unit 5's projects in each of these areas will give students a good first-approximation understanding of these issues.

Social Implications

Students learn the meaning of copyrights and patents, discuss their impact and relevance in an increasingly computationally mediated world, and learn about the simultaneous invention of the telephone.

Big Ideas 2, 3, 4, 5

The big focus of Unit 5 is on analyzing **algorithms (Big Idea 4)** and **data and information (Big Idea 3).** Students explore various ways to process, create, analyze, and visualize data through their experiments and simulations relying and building on their experience with **programming (Big Idea 5)** and **abstraction (Big Idea 2)**.

Computational Thinking P1, P2, P4, P5, P6

Students **create** several **computational artifacts (P2)** including a program for graphing data and functions, a simulation of traffic flow that lets them analyze traffic behavior on a highway, and an extension of a game from Unit 2. Students **analyze problems** (e.g., the traffic flow, list searching algorithms, and data scaling) to create suitable algorithms for solving them, and they create a timer program to help them **analyze programs (P4)** for their computational efficiency. Students continue to **connect computing (P1)** to the human experience in two contexts: through the data analysis and traffic simulation; and as they learn about copyrights, patents, and the simultaneous inventions. Students continue to **communicate (P5)** and **collaborate (P6)** through pair programming.

Enduring Understandings

- **EU 2.3** Models and simulations use abstraction to generate new understanding and knowledge.
- **EU 3.1** People use computer programs to process information to gain insight and knowledge.
- **EU 4.2** Algorithms can solve many but not all computational problems.
- **EU 5.2** People write programs to execute algorithms.
- **EU 7.1** Computing enhances communication, interaction, and cognition.
- **EU 7.2** Computing enables innovation in nearly every field.
- **EU 7.4** Computing innovations influence and are influenced by the economic, social, and cultural contexts in which they are designed and used.

Learning Objectives

- **LO 1.2.3** Create a new computational artifact by combining or modifying existing artifacts. [P2]
- **LO 1.2.5** Analyze the correctness, usability, functionality, and suitability of computational artifacts. [P4]
- **LO 2.3.1** Use models and simulations to represent phenomena. [P3]
- **LO 2.3.2** Use models and simulations to formulate, refine, and test hypotheses. [P3]
- **LO 3.1.2** Collaborate when processing information to gain insight and knowledge. [P6]
- **LO 3.1.3** Explain the insight and knowledge gained from digitally processed data by using appropriate visualizations, notations, and precise language. [P5]
- **LO 3.3.1** Analyze how data representation, storage, security, and transmission of data involve computational manipulation of information. [P4]
- **LO 4.2.1** Explain the difference between algorithms that run in a reasonable time and those that do not run in a reasonable time. [P1]
- **LO 4.2.2** Explain the difference between solvable and unsolvable problems in computer science. [P1]

- **LO 4.2.3** Explain the existence of undecidable problems in computer science. [P1]
- **LO 4.2.4** Evaluate algorithms analytically and empirically for efficiency, correctness, and clarity. [P4]
- **LO 5.2.1** Explain how programs implement algorithms. [P3]
- **LO 6.2.2** Explain how the characteristics of the Internet influence the systems built on it. [P4]
- **LO 6.3.1** Identify existing cybersecurity concerns and potential options to address these issues with the Internet and the systems built on it. [P1]
- **LO 7.1.2** Explain how people participate in a problem-solving process that scales. [P4]
- **LO 7.2.1** Explain how computing has impacted innovations in other fields. [P1]
- **LO 7.4.1** Explain the connections between computing and economic, social, and cultural contexts. [P1]

Major Activities/Requirements for Completing Unit

Activities and Learning Opportunities	**Relationship to the Enduring Understandings**	**Relationship to the Learning Objectives**
Lab 1: Algorithms • In a reversal of their number guessing game developed in Unit 2, students design an algorithm and program the computer so that the computer can guess the player's secret number, and the player tells the computer if its guesses are too high or too low. They use this as a basis for developing and analyzing list-search algorithms for ordered and unordered lists.	Building on their experiences in earlier units, students write programs to execute algorithms (**EU 5.2**) and evaluate algorithms analytically (**EU 4.2**). This lab builds toward these EUs by introducing the idea of analyzing the time requirement of an algorithm by counting operations. Students are required to develop a binary search algorithm in words and then program it.	Students develop this game by modifying their own prior work (**LO 1.2.3**). As students develop the algorithms within this program, they discuss and describe how they will be implemented (**LO 5.2.1**) and analyze the efficiency, correctness, and clarity of their algorithms (**LO 4.2.4**).

Lab 2: Graphing Data and Functions • Students create a general graphing program that plots data points on a screen whose dimensions (scale) they have determined themselves. They extend their grapher so that it can take a function as input and produce its Cartesian graph.	Students use their own graphing program to visualize data that they process to gain knowledge and insight about a situation (**EU 3.1**). They will later use it to visualize the data they get when they time their algorithms. This lab builds toward this EU by having students develop a data graphing tool as a way to gain insight into information. Students are required to program a graphing app that plots arbitrary (x, y) data from a list and can also accept a function as input and graph the values taken by the function over a range of inputs.	Students interpret and communicate the results of their data-processing by using appropriate visualizations, notations, and precise language (**LO 3.1.3**).
Lab 3: Timing Experiments • Students write a timer program that accepts a function (and inputs) and times the computation in milliseconds. Students then time several processes and use the graphing program that they built to graph the times against the size of the inputs and compare the results.	Through the development of the timer program, students continue to see that programs are written to execute algorithms (**EU 5.2**) and also build the foundation for understanding that though algorithms can solve many computational problems, there are constraints that must be taken into consideration (**EU 4.2**). This lab builds toward these EUs by having students time various algorithms, building toward the idea that there are identifiable families of algorithms with similar times as a function of input size: constant, linear, quadratic, and exponential. Students are required to build a timer procedure and apply it to	Students recognize *constant* time processes, *linear* time, *quadratic* time, and *logarithmic* time (**LO 4.2.1**) and evaluate algorithms efficiency, correctness, and clarity (**LO 4.2.4**).

	several algorithms from earlier units.	
Lab 4: Unsolvable and Undecidable Problems • The idea that some problems *do* not have solutions is credible enough to students, but the idea that some *cannot ever* have solutions, and that we can *prove* that, is harder. We introduce the notion through a slightly different idea at first. Students solve Liar/Truthteller logic puzzles to see that some are tricky but solvable and others set up inherently contradictory situations in which neither true nor false can apply. Then they see the more complex idea: problems for which truth or falsity *can* apply, but for which the state cannot (ever) be logically determined. They explore the halting problem: Can an algorithm be developed that will check *any* other algorithm to see if it will halt and return a result?	As students learn about solvable vs. unsolvable and undecidable problems, they learn that algorithms can solve many but not all computational problems (**EU 4.2**). This lab builds toward this EU by introducing students to the halting problem as an example of an unsolvable problem, sketching Turing's proof by contradiction using Snap*!* code rather than Turing Machine code. It requires students to explore and understand proof by contradiction, starting with simple Liar/Truthteller puzzles before using proof by contradiction in the context of the halting problem.	Students learn the difference between solvable and unsolvable problems (**LO 4.2.2**) and about the existence of undecidable problems (**LO 4.2.3**) through work with the halting problem.
Lab 5: Data Processing • Students import data available on the Internet, graph and analyze the data, identify questions to ask	Students use the graphing program they have developed to process and gain insights about data accessed online (**EU 3.1**). This lab builds toward this EU by using	Students collaborate as they process the data (**LO 3.1.2**) and discuss how people participate in problem-solving processes at larger scales (**LO 7.1.2**).

about the data, and process the data to answer their questions.	the graphing program from Lab 2 to analyze data retrieved from a web page, working from small examples up to a million-point data set. Students are required to answer questions about the data by using the programming tools they learned earlier to manipulate lists.	
Lab 6: Traffic Simulation • Students build a simulation to model and explore highway traffic. Students adjust various constraints (acceleration and deceleration rates, intended speed, number of cars on the road) and conduct experiments about the resulting traffic patterns.	Students use abstraction together with simulation to model and generate new understanding and knowledge about highway traffic patterns (**EU 2.3**). This lab builds toward this EU by requiring students to program a simulation (of highway traffic) so they can study how changing constraints affects the results.	Students develop a simulation to model the phenomenon of highway traffic (**LO 2.3.1**) and use their model to formulate, refine, and test hypotheses about the context (**LO 2.3.2**).
Lab 7: Copyrights and Patents • Software patents are controversial. In the US, software was generally not patentable until a 1981 Supreme Court decision. Students explore what it means to invent something, and why software might or might not be considered differently from machinery. The simultaneous invention of the telephone by several people is used as an example to debunk the apple-on-head, flash-of-insight myth. • The digital storage of information	Students consider benefits, historical contexts, and practical issues of innovations (**EU 7.1**, **EU 7.2**, **EU 7.4**). This lab builds toward these EUs by examining how the creative process is affected by laws surrounding the products of creativity: copyrights and patents, especially software patents. Students are required to struggle with their ideas about the nature of invention and the way in which creators depend on their social context, including earlier creative works. They also explore how computers and the Internet have affected the distribution of	Students discuss the effects of DRM software (**LO 1.2.5)**, the tradeoff between protecting copyrights with DRM and the inconvenience and potential loss of information that DRM entails (**LO 3.3.1**), and the tradeoff between open standards and software patents (**LO 6.2.2**). Students also consider the good or bad effects of streaming software such as BitTorrent (**LO 6.3.1**), the benefits of Creative Commons and other free licensing schemes (**LO 7.2.1**), and how technology such as region coding on DVDs affects the global digital divide

makes the marginal cost of a perfect copy zero. In principle that shouldn't affect the rights of artists, but in practice, many people feel free to copy media both for their own use and to give to friends. When classes are polled on this topic, most or all hands go up both for "who thinks it's wrong to pirate music or movies" and for "who has pirated music or movies?" This is a springboard into a class discussion of how laws have changed, how artists *should* be supported, and the use of encryption to enforce copyright.	creative works and the implications for creators.	(**LO 7.4.1**).

Outline of BJC Unit 6: Trees and Other Fractals and Unit 7: Recursive and Higher-Order Functions

Note: At the end of Unit 5, all of the CSP curriculum framework has been addressed. Units 6 and 7 contain additional material that's important to BJC, but part of Unit 6 and all of Unit 7 will come after the AP exam in May.

AP Through-Course Assessment: Create – Applications from Ideas Performance Task

- After completing Unit 6 and/or before the AP exam, students complete the through-course assessment *Create - Applications from Ideas* (12 hours in class).

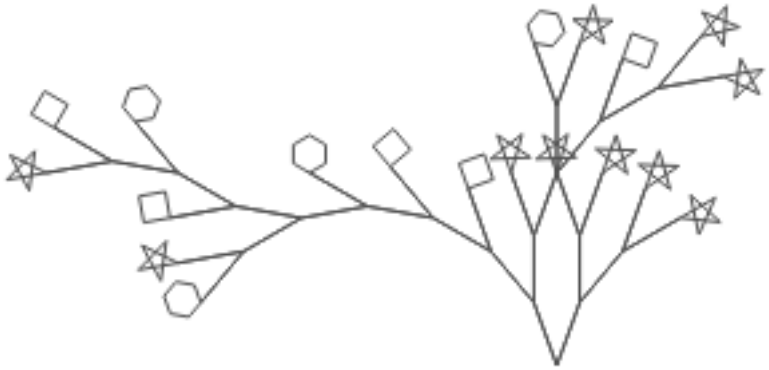
Unit 6 Description

Recursion and functional programming are two programming techniques that go beyond the Framework requirements, but are at the heart of what makes BJC unique. Unit 6 is about recursive commands, mainly fractals.

It starts with a teacher demonstration of the Vee project, in which a short program generates V shapes with randomly chosen decorations at the ends.

This same program suddenly generates arbitrarily complex results if the Vee procedure itself is added to the list of possible decorations.

This is a teacher demonstration rather than an independent lab activity because the collective gasp of the class is itself a valuable learning experience.



After the demonstration, students develop their own fractal tree program by building up from small cases (a one-level tree is just a trunk; a two-level tree is a trunk with a vee above it, etc.) so that they are not confronted at first with the seeming "cheating" of a procedure calling itself. Only after they've written several almost-identical procedures does the lab suggest combining them into one.

The students then discover why recursive procedures need a base case to terminate the recursion.

Once the tree fractal is thoroughly understood, students go on to construct other fractals (Koch snowflake, Sierpinski gasket, etc.).

The elegance of the *programs themselves* helps students see computer programs, and not just the effects they produce, as things of beauty. A key moment in developing that sense is when students understanding how a short recursive procedure can generate a deeply complex computational process.

This unit also contains a Social Implications lab exploring the effects of computers on jobs (including both the displacement of old categories of work by new ones and the on-the-job experience of workers whose output is measured by computers) and on warfare (with an emphasis on drones and why they make a qualitative difference in the political cost of war).

Unit 7 Description

Unit 7 is about recursive functions, combining the ideas of recursion, from Unit 6, and functional programming, introduced in Unit 3 with the higher order functions on lists. One highlight of the course is the implementation by students of three key list operations: `map`, `keep`, and `combine`.

After a brief introduction to the form of a recursive function (in which the recursive call is an input to a combining function, as opposed to a separate instruction as in recursive commands), we work through the example of Pascal's triangle. As part of that lab, we see how the naïve implementation takes exponential time, but techniques such as memoization can be used to create an efficient program that still maintains the essentially recursive definition of Pascal's triangle.

Other examples in the unit include conversion of numbers to and from binary, which is then generalized to arbitrary bases (up to 36, using all the letters as digits); finding the subsets of a set (a simple example of a computation that's unavoidably exponential in time, because the desired output is exponentially large); and sorting lists (using mergesort to exemplify O(n log n) algorithms, because the algorithm is simpler than Quicksort and is guaranteed n log n rather than just probabalistically n log n).

Finally, students build simple examples of recursive procedures that apply a function to every item of a list (square all the numbers, take the first letter of all the words, and so on), then *generalize that pattern* to write the `map` function, and similarly for `keep` and `combine`. This last programming lab is one highlight of the course, because it combines several central ideas: abstraction, functional programming, and recursion.