2

Big Idea 1: Creative Development

"*Writing a book is really difficult.*" —Seth Reichelson

Chapter Goals

- ➔ Inspirations for computing innovations
- ➔ Modern computing innovations
- ➔ Positive and negative effects on society, culture, or economy from computing innovations
- ➔ Hardware vs. software
- ➔ Collaboration
- ➔ Program code
- ➔ Identify inputs
- ➔ Identify outputs
- ➔ Development process
- ➔ Design a program
- ➔ Program documentation
- ➔ Program errors

Computing Innovations

Be it the flight of the vulturine guinea fowl, sonar used by whales and dolphins, or the majestic nature of termite architecture, inspiration for computing innovations can be found anywhere. People are the ones who create innovations. Computing innovations are constantly changing and building on themselves. The desire to prevent crime has created countless innovations, such as data mapping, tracking, and biometrics technologies. Not all innovations have been home runs. Many innovations have also been created that have difficulty finding an audience, such as Bluetooth toasters, smart water fountains for cats, and fridges that connect to the internet.

Advances in computing have generated and increased creativity in other fields, such as medicine, engineering, communications, and the arts. Health insurance companies are using data analytics to make decisions about which providers are more appropriate based on quantitative data. Heart monitors can take constant blood pressure and be worn as watches. Artificial legs controlled by microprocessors and automated reading applications for the blind have improved quality of life. Computing innovations can also have an impact on the arts by providing new ways to mix different types of media. Computing simulations can model real-world situations and predict outcomes based on changing variables.

Hardware Vs. Software

Hardware is the physical components of a computing device, while software is the instructions in a programming language to the computing device. A computing innovation can have hardware components. However, the computing innovation is about the software, not the hardware.

Computing hardware has gotten smaller and more powerful over the years. Moore's law predicts that the size of transistors halves every two years while the cost also halves every two years. Computers went from taking up 1,800 square feet and weighing almost 50 tons to being able to fit in your pocket.

Examples:

Software	Hardware
Operating systems	Motherboard
Driverless vehicle software to avoid crashes	Self-driving car
Dual-monitor programs for Windows	Monitor
Compiler	Transistor
Graphics card driver	Graphics card

Collaboration

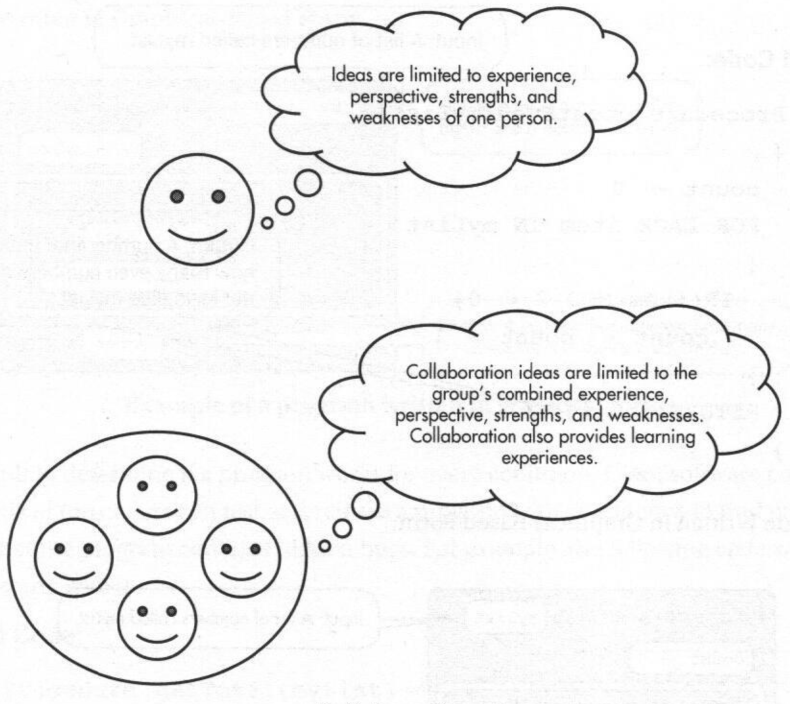
Collaboration helps people learn from each other. Collaboration that includes diverse perspectives helps to avoid bias in the development of computing innovations. For example, if females play video games at the same percentage as males, a game company might not avoid bias if it employed males to write the code for the games. Bringing in female coders could bring additional perspectives that might not have been achieved otherwise. Programming companies often hire people who not only are good programmers but also have interpersonal skills needed to collaborate effectively. Effective collaboration can help one gain insight and knowledge by applying multiple perspectives, experiences, and skill sets.

Collaboration is a learned skill. That skill includes but is not limited to:

- Communication
- Consensus building
- Conflict resolution
- Negotiation

Collaboration with others can make the programmer more self-aware. Group programming can match up your weaknesses with someone else's strengths, which results in a better product and leads to insight and knowledge not obtainable when working alone.

Collaboration facilitates the solving of computational problems by applying multiple perspectives and skill sets. Collaboration combines different individuals' resources, talents, and experiences.

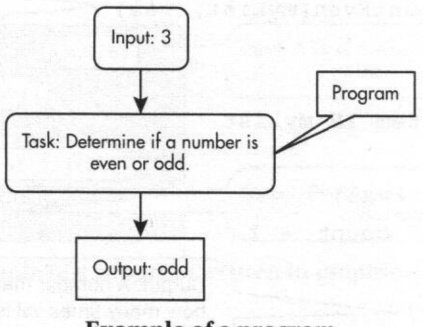

Ideas are limited to experience, perspective, strengths, and weaknesses of one person.
Collaboration ideas are limited to the group's combined experience, perspective, strengths, and weaknesses. Collaboration also provides learning experiences.

Collaboration

Collaboration is not limited by location. Current computing tools allow people in different physical locations to share data. Online collaboration tools, such as Google Docs, Zoom, Slack, Yammer, and—by the time you read this—dozens of other tools, allow programmers to collaborate from home or from anywhere that has internet access.

How Programs Function

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.



Example of a program

A code segment refers to a collection of program statements that are part of a program.

On your AP exam, code will be written in both text-based form and graphical-based form. For the following examples, this book will show both forms.

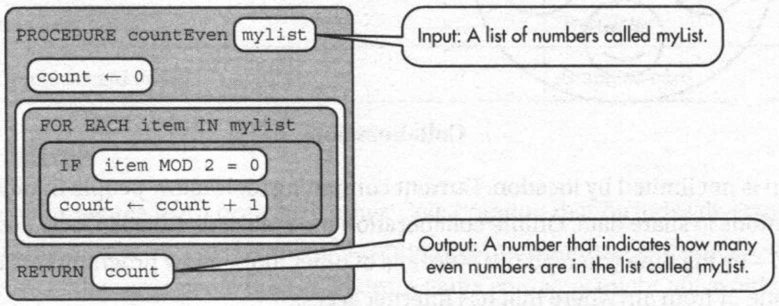
Text-Based Code:

Input: A list of numbers called myList.

```
Procedure countEven(myList)
{
 count ← 0
 FOR EACH item IN myList
 {
   IF(item MOD 2 = 0)
     count ← count + 1
 }
 RETURN(count)
}
```

Output: A number that indicates how many even numbers are in the list called myList.

Same Code Written in Graphical-Based Form:



```
PROCEDURE countEven mylist
  count ← 0
  FOR EACH item IN mylist
    IF item MOD 2 = 0
      count ← count + 1
  RETURN count
```

Input: A list of numbers called myList.
Output: A number that indicates how many even numbers are in the list called myList.

Example of a program written in graphical form

A program needs to work for a variety of inputs and situations.

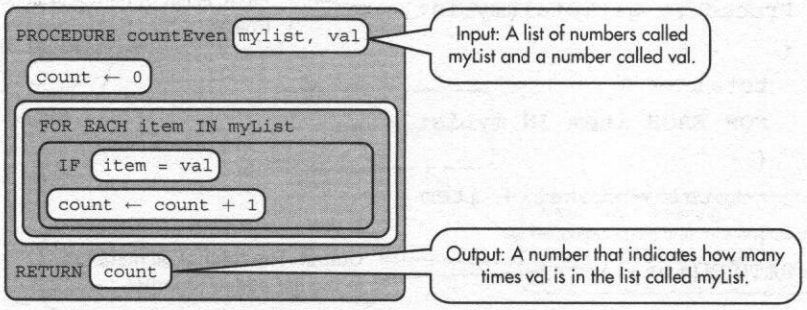
Text-Based Code:

Input: A list of numbers called myList and a number called val.

```
Procedure countEven(myList, val)
{
 count ← 0
 FOR EACH item IN myList
 {
   IF(item = val)
     count ← count + 1
 }
 RETURN(count)
}
```

Output: A number that indicates how many times val is in the list called myList.

Same Code Written in Graphical-Based Form:


Input: A list of numbers called myList and a number called val.
Output: A number that indicates how many times val is in the list called myList.

```
PROCEDURE countEven mylist, val
    count ← 0
    FOR EACH item IN myList
        IF item = val
            count ← count + 1
    RETURN count
```

Example of a program written in graphical form

It is difficult to determine if a program works for every condition. Giant software companies with hundreds of top coders can test and release a program with confidence to find out only after the release that the program contains hidden bugs. For example, the following code will work for any list that starts with 0.

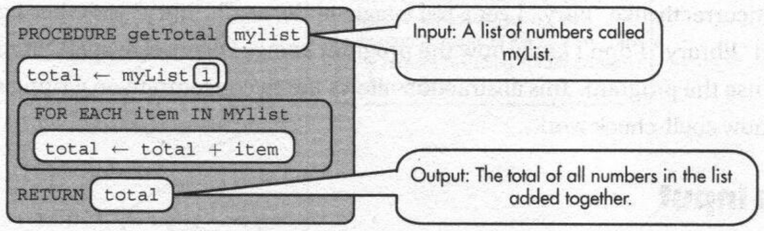
Text-Based Code:

```
Procedure getTotal(myList)
{
  total ← myList[1]
  FOR EACH item IN myList
  {
    total ← total + item
  }
RETURN(total)
}
```

Input: A list of numbers called myList.

Output: The total of all numbers in the list added together.

Same Code Written in Graphical-Based Form:



```
PROCEDURE getTotal mylist
    total ← myList 1
    FOR EACH item IN MYlist
        total ← total + item
    RETURN total
```

Example of a program written in graphical form

If myList ← {0,1,5,6} and the procedure getTotal(myList) is called, the procedure will return 12, which is the correct total.

However if myList ← {3,1,5,6} and the procedure getTotal(myList) is called, the procedure will return 18, which is not the correct total.

To correct the accumulating problem, the initial value of total must be set to 0, not myList[1].

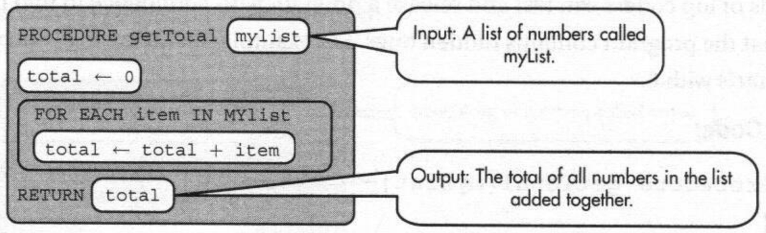
Text-Based Code:

```
Line 1:  Procedure getTotal(myList)
Line 2:  {
Line 3:    total ← 0
Line 3:    FOR EACH item IN myList
Line 4:    {
Line 5:      total ← total + item
Line 6:    }
Line 7:  RETURN(total)
Line 8:  }
```

Input: A list of numbers called myList.

Output: The total of all numbers in the list added together.

Same Code Written in Graphical-Based Form:



```
PROCEDURE getTotal mylist
  total ← 0
  FOR EACH item IN MYlist
    total ← total + item
  RETURN total
```

Example of a program written in graphical form

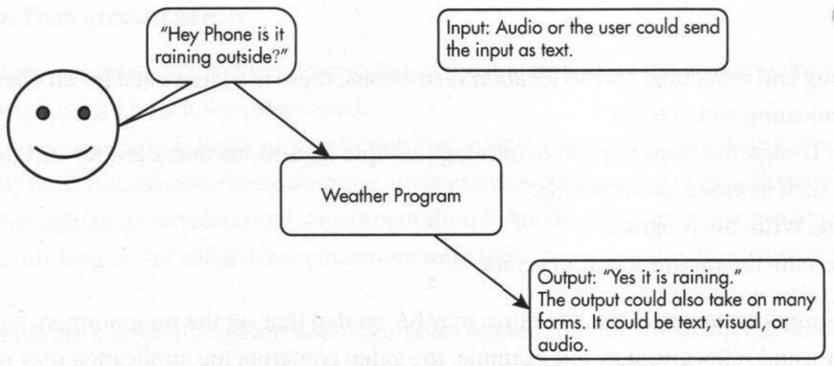
A programmer can never have too many test cases. Logical errors demonstrate why collaboration is so important in programming. Sometimes a programmer has been looking at code for so long that he or she cannot see errors. It is common for a fresh-eyed programmer to spot errors that an entrenched programmer working on code for a long time cannot see.

A complex program can be described broadly by what it does. By keeping the program abstract, the user can focus on just using the program without knowing the details of the code that makes it work. For example, right now I am typing the word "Mississippi" into a program called Microsoft Word. I have no idea how the code works, but I can use the interface. When I spell a word incorrectly like "libry," I get a red wiggle underneath "libry" with the suggestion to spell the word "library." I don't know how the program knows I wanted to spell "library," but I know how to use the program. This abstraction allows me to concentrate on writing and not worry about how spell-check works.

Program Input

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. For example, a cell phone can convert voice (audio) to text to send a message.

A weather program on your phone could take input in many forms.


"Hey Phone is it raining outside?"
Input: Audio or the user could send the input as text.
Weather Program
Output: "Yes it is raining." The output could also take on many forms. It could be text, visual, or audio.

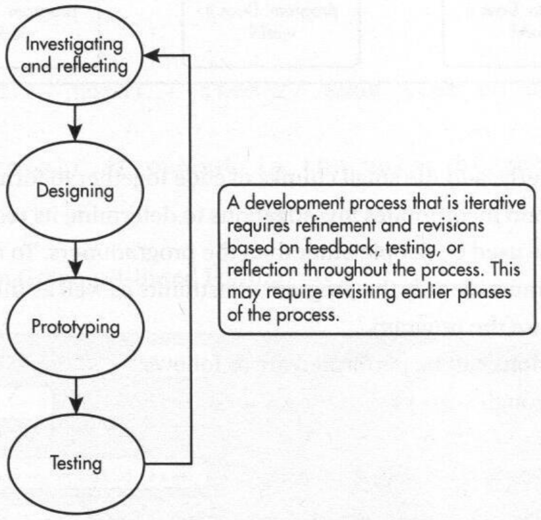
Example of input/output

This weather app was triggered by the user saying (audio) "Hey Phone. . . ," which would be an example of audio input. This triggering is called an event. The event is the action that supplies input data to a program. 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.

On your Create Performance Task, input can be any form. Mobile CSP has many sensors, such as an accelerometer, GPS, temperature, and many more that can be used as inputs.

Development Process

The development process, as shown in the figure below, can be ordered and intentional or can be exploratory in nature.


Investigating and reflecting
Designing
Prototyping
Testing
A development process that is iterative requires refinement and revisions based on feedback, testing, or reflection throughout the process. This may require revisiting earlier phases of the process.

Development process

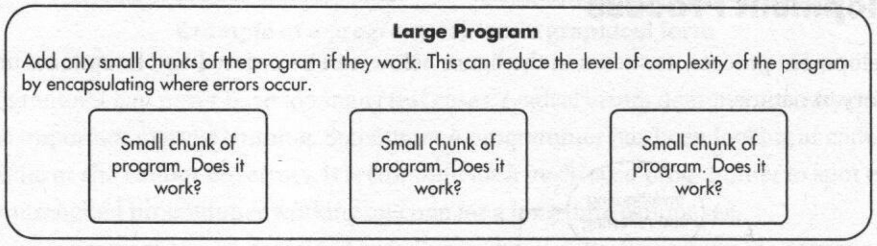
Example:

Investigating and reflecting: Due to location restrictions, there is a large need for an effective way of communicating with a team.
Designing: Design the steps needed to develop multiple algorithms that allow for certain functionalities such as video conferencing.
Prototyping: Write the program.
Testing: Execute the program to find errors.

After eliminating errors, adding features may be needed that set the programmers back to the investigating and reflecting step. For example, the video conferencing application may need to reduce background noise by adding a mute feature.

This process will be repeated until an acceptable product is reached or the programmer just runs out of time.

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. A program rarely works on the first round of development. The larger the program is, the more likely that errors will occur and the more difficult it is to find those errors. To reduce the debugging process, large programs are broken up into small steps.


Large Program
Add only small chunks of the program if they work. This can reduce the level of complexity of the program by encapsulating where errors occur.
Small chunk of program. Does it work?
Small chunk of program. Does it work?
Small chunk of program. Does it work?

Once they work properly, add the small chunks of code together to form the bigger program.

The design of a program incorporates investigations to determine its requirements. Most programs are designed to be used by people other than the programmers. To meet the needs of the users, the investigation must identify the program constraints as well as the concerns and interests of people who will use the program.

Some ways investigations can be performed are as follows:

- Collecting data through surveys
- User testing
- Interviews
- Direct observations

The design phase of a program may include:

- Brainstorming
- Planning and storyboarding
- Organizing the program into modules and functional components
- Creating diagrams that represent the layouts of the user interface
- Developing a testing strategy for the program

Program Documentation

Program documentation is a written description of the function of a code segment, event, procedure, or program and how it was developed.

Program documentation helps in developing and maintaining correct programs when working individually or in collaborative programming environments. Programmers should document a program throughout its development. Documentation helps the programmer remember what he or she was thinking or the collaborative partners were thinking at the time they were programming.

Comments are a form of program documentation written into the program that do not affect how the program runs. Comments do not affect the run speed of a program.

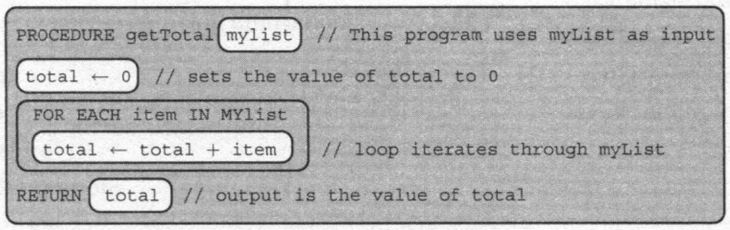
Program documentation can be used to acknowledge code segments written by someone else. This acknowledgment should include the origin or author's name. On your Create Performance Task, make sure you document if you use code segments written by someone else.

In this text, I used two backslashes // to indicate everything to the right of the two backslashes // is a comment. The following example shows how comments can appear in both text-based and graphical-based code.

Text-Based Code:

```
Procedure getTotal(myList) // this program uses myList as input
{
 total ← 0 // sets the value of total equal to 0
 FOR EACH item IN myList // loop that iterates through myList
 {
  total ← total + item // adds item to total
 }
RETURN(total) // output is the value of total
}
```

Sample Code Written in Graphical-Based Form:



```
PROCEDURE getTotal mylist // This program uses myList as input
total ← 0 // sets the value of total to 0
FOR EACH item IN MYlist
  total ← total + item // loop iterates through myList
RETURN total // output is the value of total
```

Graphical program with documentation

Documentation can look different depending on what programming language you use. On the Create Performance Task, you can either document in your code or include a separate document with your documentation.

Program Errors

They happen!

Three types of errors can occur:

- Logic error—This is a mistake in the algorithm or program that causes it to behave incorrectly or unexpectedly.
- Syntax error—This is a mistake in the program where the rules of the programming language are not followed.
- Runtime error—This is a mistake in the program that occurs during the execution of a program. Programming languages define their own runtime errors.