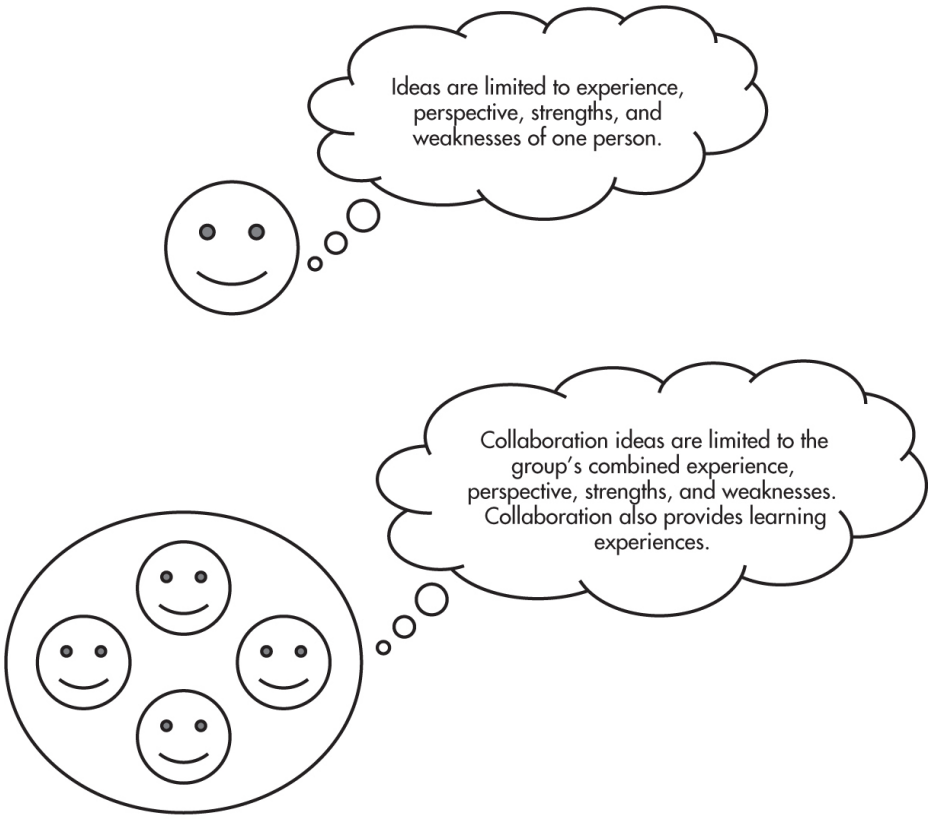
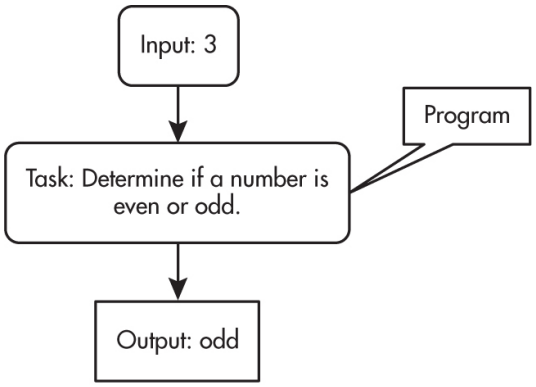


[redacted]

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.

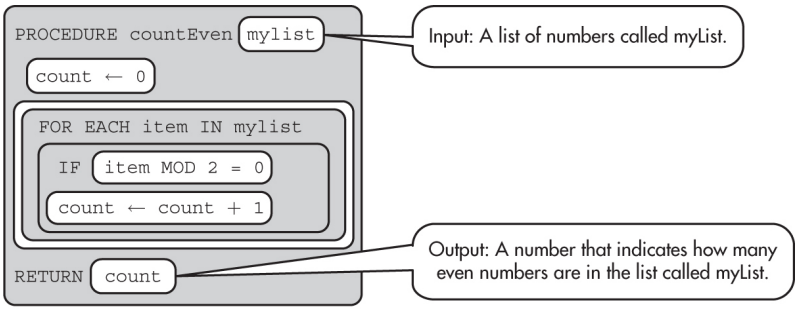
Text-Based Code:

Input: A list of numbers called myList.

```
Line 1: Procedure countEven(myList)
Line 2: {
Line 3:   count ← 0
Line 3:   FOR EACH item IN myList
Line 4:   {
Line 5:     IF(item MOD 2 = 0)
Line 6:       count ← count + 1
Line 7:   }
Line 8:   RETURN(count)
Line 9: }
```

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

Same Code Written in Graphical-Based Form:



Example of a program written in graphical form

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

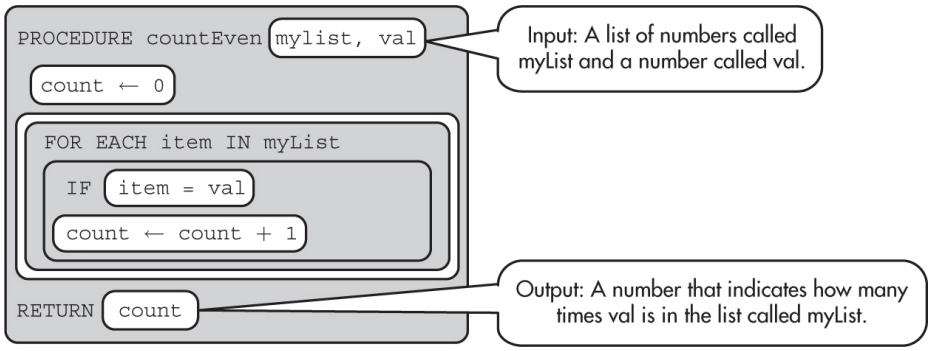
Text-Based Code:

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

```
Line 1: Procedure countEven(myList, val)
Line 2: {
Line 3:   count ← 0
Line 3:   FOR EACH item IN myList
Line 4:   {
Line 5:    IF(item = val)
Line 6:     count ← count + 1
Line 7:  }
Line 8: RETURN(count)
Line 9: }
```

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

Same Code Written in Graphical-Based Form:



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.

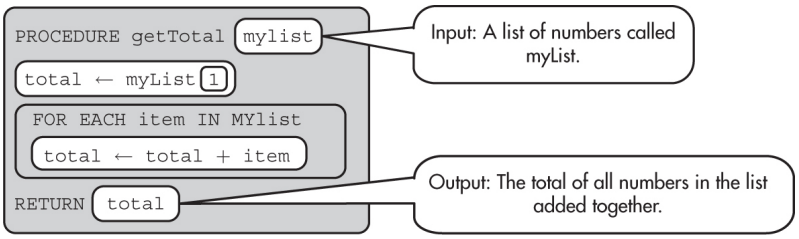
Text-Based Code:

Input: A list of numbers called myList.

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

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

Same Code Written in Graphical-Based Form:



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].

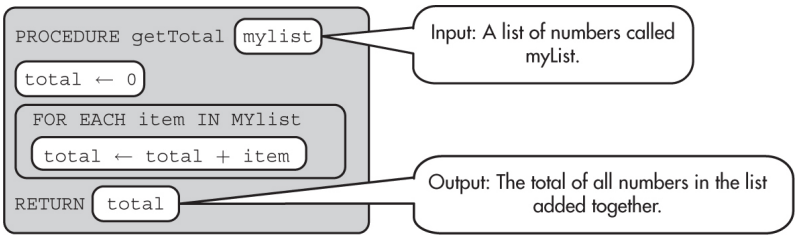
Text-Based Code:

Input: A list of numbers called myList.

```
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: }
```

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

Same Code Written in Graphical-Based Form:



Example of a program written in graphical form

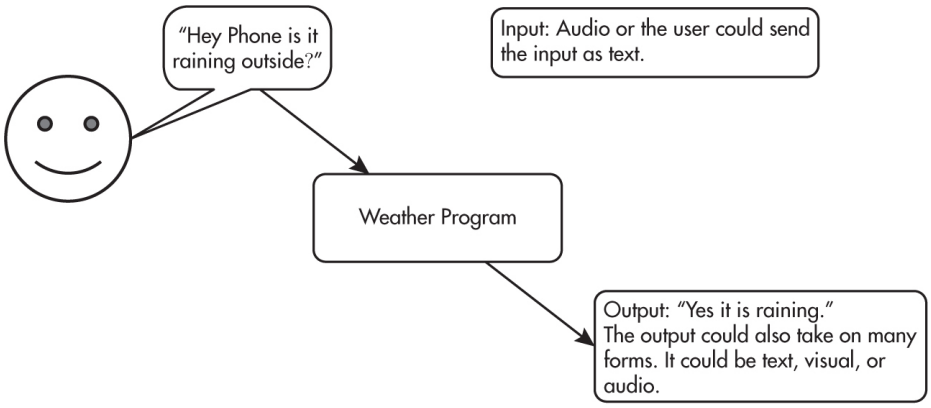
A programmer can never have too many test cases. Logical errors demonstrate why collaboration is so important in programming. Sometimes a programmer has been looking at code for so long that he or she cannot see errors. It is common for a fresh-eyed programmer to spot errors that an entrenched programmer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



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.

[redacted]