Logic, Repetition, Functions, and Input

Introduction to Programming : Logic, Repetition, Functions, and Input

Lesson 2 Overview

In this lesson, you'll learn how to create and structure decisions in your programs. You'll learn how to create loops and when to use loop-type structures in your programs. You'll learn how you can use functions, which are available in many language libraries. You'll also learn about the input validation loop and defensive programming.

2.1 Describe the use of inserting Boolean expressions for comparisons

Decision Structures and Boolean Logic

READING ASSIGNMENT

Read this assignment. Then, read Chapter 4 of your textbook.

In **decision structures**, programs can perform actions only when

certain conditions are met. You can design a computer program that makes decisions and then performs tasks accordingly. The way to program these decisions is with Boolean expressions.

With **Boolean expressions**, two scenarios are possible—one true and one false—which are indicated by a 0 or a 1. For instance, in the expression “Is this number greater than 10?” you may assign the number 0 to “yes” and 1 to “no.”

Common Boolean operators are listed in this table, although some programming languages vary in their use of Boolean operators.

BOOLEAN OPERATORS		
Operator	**Meaning**	**Description**
>	Greater than	The first value is greater than the second value.
<	Less than	The first value is less than the second value.
>=	Greater than or equal to	The first value is greater than or equal to the second value.
<=	Less than or equal to	The first value is less than or equal to the second value.
==	Equal to	The two values are equal.
!=	Not equal to	The first value isn’t equal to the second value.

If there’s only one possible outcome for a Boolean expression, use a **single-alternative decision structure**. An example is shown in Figure 4-1 of your textbook. This structure uses **If-Then** statements because the action is **conditionally executed**—it’s performed only

when a certain condition is true.

If there are two possible outcomes for a Boolean expression, use a **dual-alternative decision structure**. An example can be found in Figure 4-8 of your textbook. This structure uses **If-Then-Else** statements because if a condition is true, then the program does one thing; if the condition is false, then the program does something else.

Watch CrashCourse's video "Boolean Logic and Logic Gates" (www.youtube.com/embed/gI-qXk7XojA) for more information about this topic.

Numbers can be compared using Boolean logic, but so can strings. You can compare them to see if one is greater than or equal to another (or less than or equal to another). If you wish to test multiple conditions, you can nest one decision structure inside another. A nested decision structure is shown in Figure 4-15 of your textbook.

An alternative structure you could use in some situations is a **case structure**, which is an alternative to using a series of decisions based on a value for one variable. You might use a case structure if there are several possible values for a variable. For instance, a program fee might be based on a sliding scale according to income.

Logical operators are operators used to create complex Boolean expressions. Several compound Boolean expressions that use logical operators are used as follows:

- An **AND operator** is used in a situation in which two conditions are true, such as a worker being a full-time employee **and** having worked at the company for more than five years.
- An **OR operator** is used when you wish to execute an action if one or another condition is true. For instance, a worker is either full-time **or** the worker has worked at the company for more than five years.
- A **NOT operator** is used when an action needs to occur based on a false statement instead of a true one. For example, a worker is **not** a full-time employee and has **not** worked at the company for more than five years.

Boolean variables hold either a true or false value. They're commonly used as **flags**, or variables that signal when certain conditions exist in the program.

Make sure to answer the checkpoints in the textbook as you read the chapter. Check your answers in Appendix E at the back of your textbook.

Key Points and Links

READING ASSIGNMENT

Key Points

- In decision structures, programs can perform actions only when certain conditions are met, and the way to program these decisions is with Boolean expressions.

- With Boolean expressions, two scenarios are possible—one true and one false.
- If there's only one possible outcome for a Boolean expression, use a single-alternative decision structure; if there are two possible outcomes for a Boolean expression, use a dual-alternative decision structure.
- Numbers can be compared using Boolean logic, but so can strings.
- If you wish to test multiple conditions, you can nest one decision structure inside another.
- An alternative structure you could use in some situations is a case structure, which is an alternative to using a series of decisions based on a value for one variable.
- Logical operators are operators used to create complex Boolean expressions and include AND, OR, and NOT.
- Boolean variables are commonly used as flags, or variables that signal when certain conditions exist in the program.

Links

- CrashCourse "Boolean Logic and Logic Gates" (www.youtube.com/embed/gI-qXk7XojA)

Exercise: Decision Structures and Boolean Logic

Answer the Review Questions and the following exercises at the end of Chapter 4 in your textbook:

- **Debugging Exercise 2**
- **Programming Exercise 6**

Exercise Answer Key:

Exercise: Decision Structures and Boolean Logic

Review Questions

Multiple Choice

1. C
2. B
3. B
4. D
5. A
6. C
7. B
8. C
9. D
10. B
11. B
12. A
13. B
14. C
15. A

True or False

1. False
2. False
3. False
4. True
5. True

Short Answer

1. A conditionally executed statement is performed only when a certain condition is true.
2. A dual-alternative decision structure
3. The case structure, which is a multiple-alternative decision structure
4. The **AND** operator connects two Boolean expressions into one compound expression. Both subexpressions must be true for the compound expression to be true.
5. The **OR** operator connects two Boolean expressions into one compound expression. One or both subexpressions must be true for the compound expression to be true. It's necessary for only one of the subexpressions to be true, and it doesn't matter which.
6. The **AND** operator
7. A flag is a variable that signals when some condition exists in the program. When the flag variable is set to False, it indicates that the condition doesn't exist. When the flag variable is set to True, it means the condition does exist.

Answers for the Debugging Exercise and Programming Exercise

are in the CSC105 Lesson 2 Additional Exercise Answers supplement.

2.2 State the function of different types of loops

Repetition Structures

READING ASSIGNMENT

Read this assignment. Then, read Chapter 5 of your textbook.

Looping is a key feature of computer programming. Also called a **repetition structure**, a **loop** is an automatically repeated operation. It functions in the following way:

1. It asks a question.
2. It performs a procedure if the answer dictates it.
3. If it performed a procedure, it goes back and asks the question again.

There are two broad categories of loops:

- **Condition-controlled loops** use a true/false condition to control the number of times the statement repeats. These include:
    - **While loops**, which perform a task while a condition is true or test its condition before performing an iteration (Figure 5-1 of your textbook)
    - **Do–While loops**, in which statements are always run at

least once before the condition is tested (Figure 5-7 of your textbook)
    - **Do–Until loops**, in which statements run until a condition is false
- **Count-controlled loops** repeat a specific number of times and include the **For statement** that performs the following three actions:
    - **Initialization**: The counter variable is initialized to a starting value that's dependent on the situation before the loop begins.
    - **Test**: The loop tests the counter variable by comparing it to a maximum value. If it’s less than or equal to the maximum value, the loop iterates. If it’s greater than the maximum value, the loop ends.
    - **Increment**: The loop **increments** (increases) the counter variable by adding 1 to it.

You also can use a loop to **add**, or accumulate, totals. You would use a variable called an **accumulator**, which is like a counter, except that it can add a value greater than one. (With a counter, you can increment only by one.) Likewise, you can **decrement**, or subtract from, the variable. You can also control looping with a **variable sentinel value**, which means that the value you’re using in your loop can change.

A loop within a loop is called a **nested loop**. The loop that’s within another loop is called the **inner loop**; the outer one is called the **outer**

loop.

Make sure to answer the checkpoints in the textbook as you read the chapter. Check your answers in Appendix E at the back of your textbook.

Key Points and Links

READING ASSIGNMENT

Key Points

- Also called a repetition structure, a loop is an automatically repeated operation.
- There are two broad categories of loops: condition-controlled loops that use true/false conditions (**While** loops, **Do-While** loops, and **Do-Until** loops) and count-controlled loops that repeat a specific number of times (**For** statement).
- You also can use a loop to accumulate or decrement totals by using a variable called an accumulator, which is like a counter except that it can add a value greater than one.
- You can control looping with a variable sentinel value, which means that the value you're using in your loop can change.
- A loop within a loop is called a nested loop.

Exercise: Repetition Structures

Answer the Review Questions and the following exercises at the end of Chapter 5 in your textbook:

- **Debugging Exercise 2**
- **Programming Exercise 1**

Exercise Answer Key:

Exercise: Repetition Structures

Review Questions

Multiple Choice

1. B
2. D
3. D
4. A
5. B
6. A
7. C
8. B
9. D
10. A

True or False

1. False
2. True
3. False
4. True
5. False
6. False

7. True
8. False
9. True
10. False

Short Answer

1. By indenting the statements in the body of the loop, you visually set them apart from the surrounding code. This makes your program easier to read and debug. Also, this is similar to the style that most programmers follow when writing loops.
2. A pretest loop tests its condition before each iteration. A posttest loop tests its condition after each iteration.
3. A condition-controlled loop uses a true/false condition to control the number of times that it repeats.
4. A count-controlled loop repeats a specific number of times.
5. Initialization, test, and increment
6. An infinite loop continues to repeat until the program is interrupted. Infinite loops usually occur when the programmer forgets to write code inside the loop that makes the test condition false. Here's an example of pseudocode that contains an infinite loop:

   x = 99
   While x > 0
   Display x
   End While
7. A **While** loop

8. If the accumulator starts with any value other than 0, it won't contain the correct total when the loop finishes.
9. You can write a loop that processes a list of data items when you don't know the number of data items in the list, and (1) you don't want to ask the user if there are more items at the end of each loop iteration, and (2) you don't want to require the user to know the number of items in the list in advance.
10. A sentinel value must be unique enough that it won't be mistaken as a regular value in the list.

Answers for the Debugging Exercise and Programming Exercise are in the CSC105 Lesson 2 Additional Exercise Answers supplement.

2.3 Explain the use of functions in programming languages

Functions

READING ASSIGNMENT

Read this assignment. Then, read Chapter 6 of your textbook.

You first learned about modules in Section 1.3. Now you'll learn about a specific type of module called a **function**, which returns a value back to the program part that called it.

Functions are useful in certain situations, such as when you want the

user of your program to input data. For instance, perhaps your program requires the user's age to perform a calculation. A function would be useful for getting the data from the user and performing whatever calculation you program.

For more information about functions, watch CrashCourse's video, "Programming Basics: Statements & Functions." (www.youtube.com/embed/l26oaHV7D40)

Many functions are built into programming languages so that you can call them at any time. These prewritten functions are called **library functions**. For instance, the **random** function generates a random number, such as a computer game in which a player rolls dice.

Common library functions include the following:

- **Mathematical functions**, which typically accept one or more values as arguments, perform a mathematical operation using the arguments and return the result
- **Data type conversion functions**, which convert values from one data type to another, such as a real number to an integer
- **Formatting functions**, which format numbers in some way, such as numbers to currency amounts
- **String functions**, which include returning the length of a string, accepting strings as arguments, converting the case of strings, returning substrings, and more

You can write your own functions in most programming languages.

You can write functions to return numbers, strings of characters, or Boolean values. To write a function, you need to define its important characteristics:

- The **header** specifies the data type of the value that's returned from the function, the name of the function, and any parameter variables used by the function to accept arguments.
- The **body** is comprised of one or more statements that are executed when the function is called.
- A **Return statement** specifies the value that's returned from the function when the function ends.

Make sure to answer the checkpoints in the textbook as you read the chapter. Check your answers in Appendix E at the back of your textbook.

Key Points and Links

READING ASSIGNMENT

Key Points

- A function is a specific type of module that returns a value back to the program part that called it.
- Functions are useful in certain situations, such as when you want the user of your program to input data.
- Library functions are prewritten functions built into programming languages so that you can call them at any time.
- Common library functions include mathematical functions, data-

type conversion functions, formatting functions, and string functions.

- You can write your own functions to return numbers, strings of characters, or Boolean values in most programming languages.

Links

- CrashCourse "Programming Basics: Statements & Functions" (www.youtube.com/embed/l26oaHV7D40)

Exercise: Functions

Answer the Review Questions and the following exercises at the end of Chapter 6 in your textbook:

- **Algorithm Workbench 1, 2, and 4**
- **Debugging Exercise 1**
- **Programming Exercise 7**

Exercise Answer Key:
Exercise: Functions

Review Questions

Multiple Choice

1. B
2. D
3. A

4. C
5. D
6. B
7. C
8. B
9. A
10. A

True or False

1. False
2. True
3. True
4. True
5. False

Short Answer

1. A function returns a value back to the part of the program that called it, and a module doesn't.
2. The function's input, processing, and output
3. It's thrown away.
4. A string inside another string
5. To convert a string to an Integer or a Real value
6. To test a string and determine whether the string can be converted to an Integer or Real

Answers for the Algorithm Workbenches, Debugging Exercises,

and Programming Exercises are in the CSC105 Lesson 2 Additional Exercise Answers supplement.

2.4 Identify the process of creating an input validation loop

Input Validation

READING ASSIGNMENT

Read this assignment. Then, read Chapter 7 of your textbook.

Computer programmers use the phrase “garbage in, garbage out” (**GIGO**) in reference to the fact that computers can’t differentiate between good and bad data. To try to prevent bad data from being entered, programmers use **input validation**, which refers to testing data to make sure it’s valid.

One way to test data is to create an **input validation loop** that repeats as long as the data that’s entered is bad data. For example, suppose a payroll program has a maximum of 40 hours a week per employee. If a user enters a three-digit number, such as 100 hours instead of 10 hours, the data would be rejected.

To test input **before** beginning a loop, you would create a **priming read**, which aims to get the first input value to be tested in the validation loop. You could also use a posttest loop to validate input, but that wouldn’t display an error message, so it’s not the preferred

method.

Input validation is a type of **defensive programming**, which aims to prevent potential errors from happening by designing a program to avoid those errors. This is especially helpful for errors that aren't so obvious, such as the **empty input** error. This error occurs when an **Input** statement executes, and the user simply presses the **Enter** key without typing a value. Another often overlooked error is the entry of the wrong type of data.

Make sure to answer the checkpoints in the textbook as you read the chapter. Check your answers in Appendix E at the back of your textbook.

Key Points and Links

READING ASSIGNMENT

Key Points

- Computer programmers use the phrase "garbage in, garbage out" (GIGO) in reference to the fact that computers can't differentiate between good and bad data.
- Programmers use input validation to try to prevent bad data from being entered.
- One way to test data is to create an input validation loop that repeats as long as the data that's entered is bad data.
- To test input before beginning a loop, you would create a priming read, which aims to get the first input value to be tested in the

validation loop.

- Input validation is a type of defensive programming, which aims to prevent potential errors from happening by designing a program to avoid those errors.

Exercise: Input Validation

Answer the Review Questions and the following exercises at the end of Chapter 7 in your textbook:

- **Algorithm Workbench 4 and 5**
- **Debugging Exercises 1 and 2**
- **Programming Exercise 1**

Exercise Answer Key:
Exercise: Input Validation

Review Questions

Multiple Choice

1. B
2. C
3. D
4. A
5. B

True or False

1. False
2. False
3. False

Short Answer

1. It refers to the fact that computers can't tell the difference between good data and bad data. If a program's user provides bad data as input, then the program will process that bad data, and as a result, will produce bad data as output.
2. When input is given to a program, it should be inspected before it's processed. If the input is invalid, then it should be discarded, and the user should be prompted to enter the correct data.
3. The purpose of the priming read is to get the first input value that will be tested by the validation loop.
4. Using a posttest loop, as shown in this chapter, doesn't display an error message when invalid data is entered. It simply repeats the original prompt each time the loop iterates. This might be confusing to the user, so it's usually better to have a priming read followed by a pretest validation loop.

Answers for the Algorithm Workbenches, Debugging Exercises, and Programming Exercise are in the CSC105 Lesson 2 Additional Exercise Answers supplement.

2.5 Design a program with validation functions

CSC105 Graded Project 2

READING ASSIGNMENT

Your project must be submitted as a zipped/compressed (*.zip) file that includes the following files:

- Text file (*.txt) of your pseudocode
- Screenshot(s) in JPEG format (*.jpg) of your flowchart
- A Rich Text Format (*.rtf) or Microsoft Word (*.doc) file that lists the following information:
    - Your name
    - Your student ID number
    - The exam number
    - Your email address

For information on how to take and save a screenshot on your computer or to zip files, review the instructions for CSC105 Graded Project 1. Save your compressed file as **[Your Name]_CSC105_GradedProject2.** Your project will be individually graded by your instructor and therefore may take up to five to seven days to grade. To submit your graded project, follow these steps:

- Log into your student portal.
- Click **Take Exam** next to the lesson you're working on.
- Find the exam number for your project at the top of the Project Upload page.
- Follow the instructions provided to complete your exam.

Be sure to keep a backup copy of any files you submit to the school!

Introduction

You'll apply the concepts of Lesson 2 to design a program with

validation functions.

Instructions

You'll create both pseudocode and a flowchart to design a program that asks for fat grams and calories in a food item.

Validate the input as follows:

- Make sure the numbers of fat grams and calories aren't less than 0.
- Ensure that the number of calories entered isn't greater than **fat grams × 9.**

Once correct data has been entered, the program should calculate and display the percentage of calories that come from fat. Use the following formula:

Percentage of calories from fat = (Fat grams × 9) ÷ calories

Some nutritionists classify a food as "low fat" if less than 30 percent of its calories come from fat. If the results of this formula are less than 0.3, the program should display a message indicating the food is low in fat.

Review Appendices B and C in your textbook for guidance when working on your project. Use free trials of any of the programs listed in CSC105 Graded Project 1 to create the flowchart. Write your pseudocode in a plain-text editor such as Notepad or TextEdit, and

save as a text file (*.txt). Save a screenshot of your flowchart as a JPEG file (*.jpg).

Grading Criteria

Your instructor will use the following guidelines to grade your project.

The program is designed to effectively process an input of data, ask for fat grams and calories in a food item, and display a message indicating a food is low in fat.	34 points
The pseudocode is accurate and in the correct format.	33 points
The flowchart is accurate and uses the appropriate shapes.	33 points
TOTAL	**100 points**

Lesson 2 Review

Self-Check

1. In a flowchart, the _____ symbol indicates that some condition must be tested.

a. parallelogram

b. oval

c. rectangle

d. diamond

2. Which two logical operators perform short-circuit evaluation?

a. **OR** and **BUT**

b. **AND** and **NOT**

c. **AND** and **OR**

d. **OR** and **NOT**

3. Which of the following expressions would check if a number, x, is between 90 and 100, inclusive?

a. **x >= 90 OR x <= 100**

b. **x > 90 OR x < 100**

c. **x >= 90 AND x <= 100**

d. **x > 90 AND x < 100**

4. What type of operator determines whether a specific relationship exists between two values?

a. Boolean

b. Relational

c. Logical

d. Mathematical

5. Which of the following operators is used in most languages to test if two operands don't have the exact same value?

a. **==**

b. **=!**

c. **!=**

d. **!**

6. Which type of loop is specifically designed to initialize, test, and

increment or decrement a counter variable?

a. **While**

b. **For**

c. **Do-Until**

d. **Do-While**

7. How many times would the following loop iterate? **For j = 1 To 5**
Step 2
Display j
End For

a. 0

b. 5

c. 4

d. 3

8. Statements that appear between the **While** and **End While** statements are said to compose the

a. **While** statements.

b. iteration values.

c. loop.

d. loop body.

9. A/An _____ represents a special value that marks the end of a list of values.

a. End For

b. Step

c. counter

d. sentinel

10. The following is an example of a _____ loop. **While x < 10**

Set sum = sum + x

Set x = x + 1

End While

a. **Do-While**

b. **While**

c. condition-controlled

d. count-controlled

11. Which function joins two strings?

a. **substring**

b. **contains**

c. **concatenate**

d. **append**

12. What's the value of y after the following statement is coded and executed, given that x = 25? **Set y = sqrt(x)**

a. 125

b. 25

c. 5

d. 0

13. What would display if the following pseudocode were coded and executed? **Declare String str1 = "car"**

Declare String str2 = "green"

Set message = append(str1, str2)

Display "You have a " , message

a. You have a car green

b. You have a greencar

c. You have a green car

d. You have a cargreen

14. What would display if the following pseudocode were coded and executed? **Declare String user = "Martha and George"**

Declare Integer number

Set number = length(user)

Display number

a. 15

b. 19

c. 17

d. Martha and George

15. A function _____ comprises one or more statements that are executed when the function is called.

a. body

b. data type

c. header

d. definition

16. A/An _____ is another term for input validation.

a. type mismatch

b. input error

c. input trap

d. error trap

17. Which function could be used to validate an entry for a user's chosen name that must be between 4 and 12 characters?

a. **length**

b. **isString**

c. **random**

d. **toUpper**

18. What would display if the following statements are coded and executed and the user enters **-3** at the first prompt, **0** at the next prompt, and **22** at the third prompt? **Display "Enter your age:"**

Input age

While age <= 0

Display "Impossible! Enter an age greater than 0:"

Input age

End While

Display "Thank you."

a. Impossible! Enter an age greater than 0:
Impossible! Enter an age greater than 0:
Impossible! Enter an age greater than 0:

b. Impossible! Enter an age greater than 0:
Impossible! Enter an age greater than 0:
Thank you.

c. Impossible! Enter an age greater than 0:
Thank you.

d. Thank you.

19. The priming read is placed _____ the loop.

a. inside

b. below

c. before and inside

d. before

20. Designing a program to avoid common errors is called _____ programming.

a. detection
b. direct
c. defensive
d. validation

Self-Check Answer Key

1. diamond
   Explanation: In a flowchart, the diamond symbol indicates that some condition must be tested.
   Reference: Section 2.1

2. **AND** and **OR**
   Explanation: **AND** and **OR** are two logical operators that perform short-circuit evaluation.
   Reference: Section 2.1

3. **x >= 90 AND x <= 100**
   Explanation: The expression **x >= 90 AND x <= 100** would check if a number, **x**, is between 90 and 100, inclusive.
   Reference: Section 2.1

4. Relational
   Explanation: A relational operator determines whether a specific relationship exists between two values.
   Reference: Section 2.1

5. **!=**

   Explanation: The **!=** operator is used in most languages to test if two operands don't have the exact same value.

   Reference: Section 2.1

6. **For**

   Explanation: The **For** loop is specifically designed to initialize, test, and increment or decrement a counter variable.

   Reference: Section 2.2

7. 3

   Explanation: The following loop would iterate three times: **For j = 1 To 5 Step 2**

   Display j

   End For

   Reference: Section 2.2

8. loop body.

   Explanation: Statements that appear between the **While** and **End While** statements are known as the loop body.

   Reference: Section 2.2

9. sentinel

   Explanation: A sentinel represents a special value that marks the end of a list of values.

Reference: Section 2.2

10. count-controlled

    Explanation: The following is an example of a count-controlled loop: **While x < 10**

    Set sum = sum + x

    Set x = x + 1

    End While

    Reference: Section 2.3

11. **concatenate**

    Explanation: The **concatenate** function joins two strings.

    Reference: Section 2.3

12. 5

    Explanation: Given that x = 25, the value of y is **5** if the following statement is coded and executed: **Set y = sqrt(x)**

    Reference: Section 2.3

13. You have a cargreen

    Explanation: The phrase **You have a cargreen** would display if the following pseudocode were coded and executed: **Declare String str1 = "car"**

    Declare String str2 = "green"

    Set message = append(str1, str2)

    Display "You have a " , message

Reference: Section 2.3

14. 17

    Explanation: If the following pseudocode were coded and executed, **17** would display: **Declare String user = "Martha and George"**
    Declare Integer number
    Set number = length(user)
    Display number
    Reference: Section 2.3

15. body

    Explanation: A function body comprises one or more statements that are executed when the function is called.
    Reference: Section 2.4

16. error trap

    Explanation: An error trap is another term for input validation.
    Reference: Section 2.4

17. **length**

    Explanation: The **length** function could be used to validate an entry for a user's chosen name that must be between 4 and 12 characters.
    Reference: Section 2.4

18. Impossible! Enter an age greater than 0:
    Impossible! Enter an age greater than 0:
    Thank you.
    Explanation: The following would display if the statements were coded and executed and the user entered **-3** at the first prompt, **0** at the next prompt, and **22** at the third prompt: **Impossible! Enter an age greater than 0:**
    Impossible! Enter an age greater than 0:
    Thank you.
    Reference: Section 2.4

19. before
    Explanation: The priming read is placed before the loop.
    Reference: Section 2.4

20. defensive
    Explanation: Designing a program to avoid common errors is called defensive programming.
    Reference: Section 2.4

Flash Cards

1. Term: Decision Structure

Definition: Variables that signal when certain conditions exist in the program

2. Term: Single-Alternative Decision Structure
Definition: When only one alternative path of execution exists

3. Term: Dual-Alternative Decision Structure
Definition: When two possible paths of execution exist

4. Term: Function
Definition: A module that returns a value back to the program part that called it

5. Term: Accumulator
Definition: A counter that can add a value greater than one

6. Term: Repetition Structure
Definition: A loop; an automatically repeated operation

7. Term: Condition-Controlled Loop
Definition: Uses a true/false condition to control the number of times a statement repeats

8. Term: Count-Controlled Loop
Definition: Repeats a statement a specific number of times

9. Term: Function Header
Definition: Specifies (1) the data type of the value that's returned from the function, (2) the name of the function, and (3) any parameter

variables used by the function to accept arguments

10. Term: Function Body
Definition: Comprised of one or more statements that are executed when the function is called

11. Term: Data Type Conversion Functions
Definition: Change values from one data type to another

12. Term: Input Validation
Definition: Testing data to make sure it's valid

13. Term: Defensive Programming
Definition: Aims to prevent potential errors from happening by designing a program to avoid those errors

14. Term: Priming Read
Definition: Aims to get the first input value to be tested in the validation loop

15. Term: Input Validation Loop
Definition: Repeats as long as the data that's entered is bad data