CHAPTER

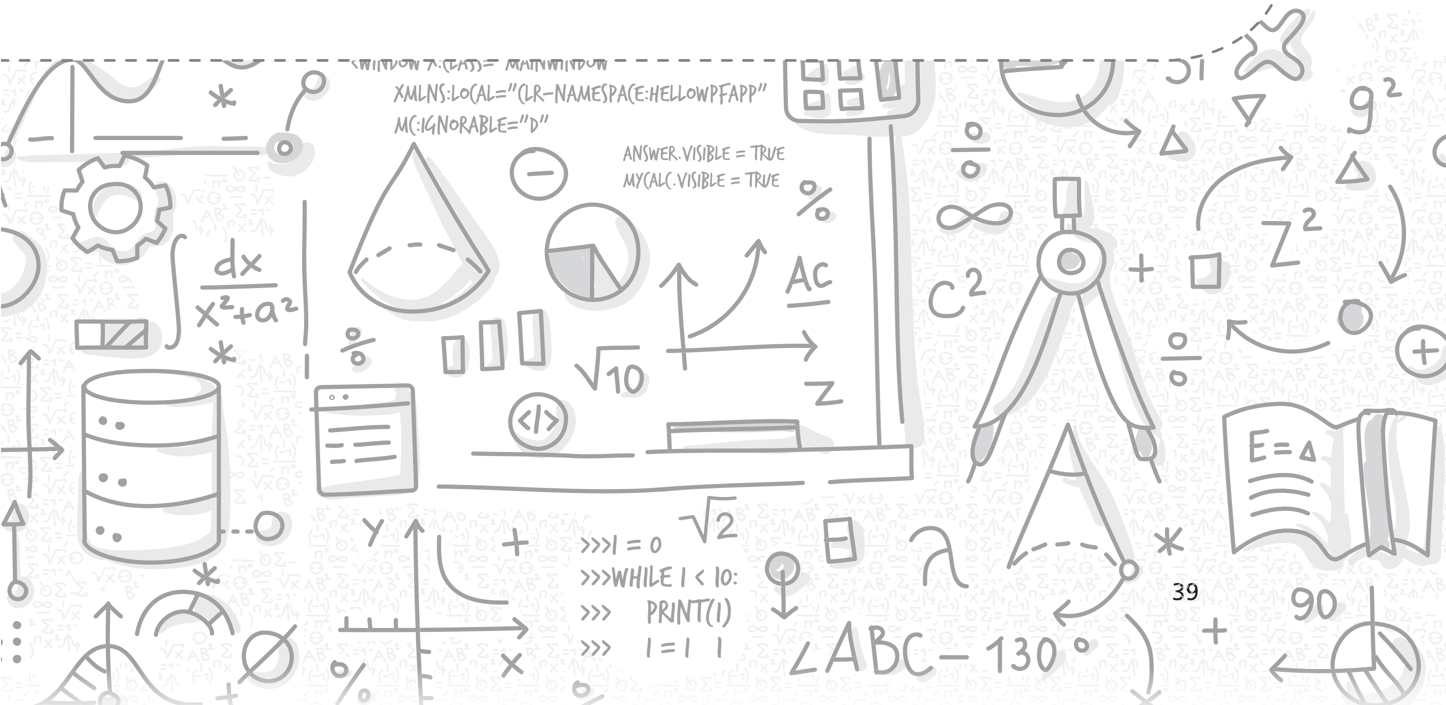
3

Data Dialogue

This chapter explores the fundamental concept of data and variables as they relate to coding. Definitions and examples are shared along with a project to explore variables.

Also included in this chapter:

- **Definitions of data types and how they are declared in different programming languages**
- **Standards-aligned variable project using Visual Studio**
- **Feature: “Coding + Math during Preservice Teacher Development” by Dr. Cory Gleasman**
- **Resources for learning more about data and variables**

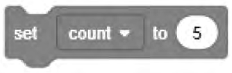
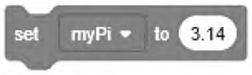
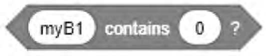
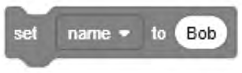


Given the importance of data as one of the foundational building blocks of any program, it is important that beginning programmers have a solid foundation in how the variables that hold data are created and manipulated in code. Even the best of programming efforts can be derailed by errors related to using variables that are of the incorrect, or incompatible, types for the data being stored in them. However, before we discuss the variables and their types, a quick primer on how computers actually store data, any kind of data, will be helpful in shaping overall understanding of how computer programs work.

All data on computers are represented as a series of binary numbers stored in bits. Binary numbers have only one of two values: 0 or 1. A *bit* is the smallest unit of information storage on a computer; however, the smallest unit that can be given an address by a computer is a *byte*, which is comprised of eight bits. Although having students learn the base-2 system of mathematics is not particularly useful for learning programming, they should at least understand that each data type has a limited number of bytes allocated to it, as well as restrictions on the kinds of data each can store. It needs to be understood that once a section of memory is allocated to a variable, that space cannot be used for any other purpose within that program for the duration it is running. Given that computer memory is a limited resource, it is not possible to allocate unlimited memory to variables, and if too much space is allocated, the increased use of memory runs the risk of slowing the program's execution. Thus, programming languages limit the amount of resources, or bytes, allocated to different types of variables and rely on the programmer to select those appropriate to their program's needs. Beyond memory needs, there is another, more important, reason for selecting the appropriate type of variable. When your program attempts to perform some kind of operation on the data, mistyped data (i.e., data of one type being assigned to a variable of a different type) can lead to program crashes, lost data, or other unexpected results.

There are many variable types across the different programming languages, but most programming tasks that beginning programmers can reasonably expect to encounter can be handled with a small subset of data types that handle the most common forms of data. In a K–5 setting, data in the form of integers, floating decimals, Boolean values, and strings should handle the vast majority of their data storage needs. The integer data type is capable of storing numeric values from about negative two billion to positive two billion, which is more than sufficient to handle a wide range of programming possibilities. The float data type provides even greater numeric data capacity and includes accuracy to seven digits beyond the

Table 3.1 **Declaring Basic Data Types by Programming Language**

TYPE	INTEGER	FLOATING DECIMAL	BOOLEAN	STRING
C	int myCount = 5;	float myPi = 3.14;	int myB1 = 0;	char name[10] = “Bob”;
C#	int myCount = 5;	float myPi = 3.14f;	bool myB1 = false;	string name = “Bob”;
Python	myCount = 5	myPi = 3.14	myB1 = False	name = “Bob”
Visual Basic	Dim myCount as Integer = 5	Dim myPi as float = 3.14	Dim myB1 as Boolean = 0	Dim name as String = “Bob”
Java	int myCount = 5;	float myPi = 3.14f	Boolean myB1; myB1 = false;	String name = “Bob”;
Scratch				

decimal point for fractional values. Programs using fractional numbers written by elementary students would not likely require more precision than a float variable can provide. The frequent use of Boolean logic in programming, testing whether some condition is true or false, makes the Boolean variable type important to understand as an early foundational CT concept as well. Finally, all programs will use the string variable type in some capacity, which stores alphanumeric characters used to communicate with the end user. String variables are stored as text and have a capacity of about one billion characters.

Before using a variable in code, the programmer must *declare* the variable to tell the computer to reserve space for the data that it will hold. In programming languages that are *strongly typed,* the programmer must also specify the type of data that will be stored in the variables at the time they declare them. Table 3.1 provides examples of how the previously described subset of variable types are declared across various languages. Note that Python and Scratch are dynamically typed languages; they set variable types by attempting to interpret the data assigned to them. While this makes using variables quick and easy, it can lead to errors when the programmer crosses types in the handling of data. It also does not require the programmer to consider variable types, which is a skill useful and necessary to work with some other languages. Scratch is the easiest to declare of the examples provided, but it does not require any syntax and its programs only run on the Scratch website. Text-based programming languages require syntax in order to allow the code to run in other platforms and contexts.

In the examples provided in Table 3.1, the terms beginning with “my” are variables that are created and named by the programmer; hence they could be named anything as long as they follow the guidelines for the particular language. In text-based languages, they generally can contain letters, digits, and the underscore character (_). They are case-sensitive (i.e., “score” is different from “Score”) and often are required to begin with a letter or an underscore. In addition, text-based programming languages each have a set of keywords that are reserved for a particular purpose (such as *int, float*, and *bool*), and as such, they cannot be used for variable names. Fortunately, many application program interface (API) designs include features that alert the programmer when variable naming conventions aren’t followed.

Getting Out the Blocks

Project: Introduction to Variables

k12stemequity.com/intro-to-variables

Overview

Although instruction on how variables are declared, named, and typed can occur offline in unplugged activities, we encourage allowing students to experiment with creating variables in a computer API so they gain familiarity in an authentic development context. For our project example, we'll use the C# (pronounced "C Sharp") programming language to allow its strongly typed behavior to highlight the differences in how data are handled in code. We will use the Microsoft Visual Studio (MVS) Community Edition (available for free download for PC or Mac at visualstudio.microsoft.com/downloads) to take advantage of its built-in debugging and coding assistance. For this sample lesson, you will need to launch a new project from the MVS file menu and create a Console App using the C# language (see instructions that follow).

Duration

Depending on your students' familiarity with basic coding concepts and Microsoft Visual Studio, they will need approximately forty-five minutes to an hour to complete this project. Allow students to work in pairs when necessary. Consider separating this project into two lessons and give students time on the second day to discuss any obstacles or difficulties encountered while working through their projects.

Standards Addressed

CSTA Standards for CS Educators

- **5c.** Promote student self-efficacy. Facilitate students' engagement in the learning process and encourage students to take leadership of their own learning by encouraging creativity and use of a variety of resources and problem-solving techniques.
- **5e.** Encourage student communication about computing. Create meaningful opportunities for students to discuss, read, and write about computing.

ISTE Standards for Students

- **5.** Computational Thinker. Students develop and employ strategies for understanding and solving problems in ways that leverage the power of technological methods to develop and test solutions.
- **5b.** Students collect data or identify relevant data sets, use digital tools to analyze them, and represent data in various ways to facilitate problem-solving and decision-making.

CSTA K–12 Standards

ALGORITHMS AND PROGRAMMING

- **1B-AP-09.** Create programs that use variables to store and modify data.
- **2-AP-11.** Create clearly named variables that represent different data types and perform operations on their values.
- **3A-AP-14.** Use lists to simplify solutions, generalizing computational problems instead of repeatedly using simple variables.

K–12 Computer Science Framework

- **Practice 4.** Developing and using abstractions
- **Practice 5.** Creating computational artifacts

CCSS Mathematical Practices

- **MP.1.** Make sense of problems and persevere solving them.
- **CCSSM 2.OA.1.** Represent and solve problems involving addition and subtraction. Use addition and subtraction within 100 to solve one- and two-step word problems involving situations of adding to, taking from, putting

together, taking apart, and comparing, with unknowns in all positions (e.g., by using drawings and equations with a symbol for the unknown number to represent the problem).

- **3.OA.8.** Solve problems involving the four operations, and identify and explain patterns in arithmetic. Solve two-step word problems using the four operations. Represent these problems using equations with a letter standing for the unknown quantity. Assess the reasonableness of answers using mental computation and estimation strategies, including rounding.
- **4.OA.3.** Use the four operations with whole numbers to solve problems. Solve multistep word problems posed with whole numbers and having whole-number answers using the four operations, including problems in which remainders must be interpreted. Represent these problems using equations with a letter standing for the unknown quantity. Assess the reasonableness of answers using mental computation and estimation strategies, including rounding.

Brennan and Resnick's Framework

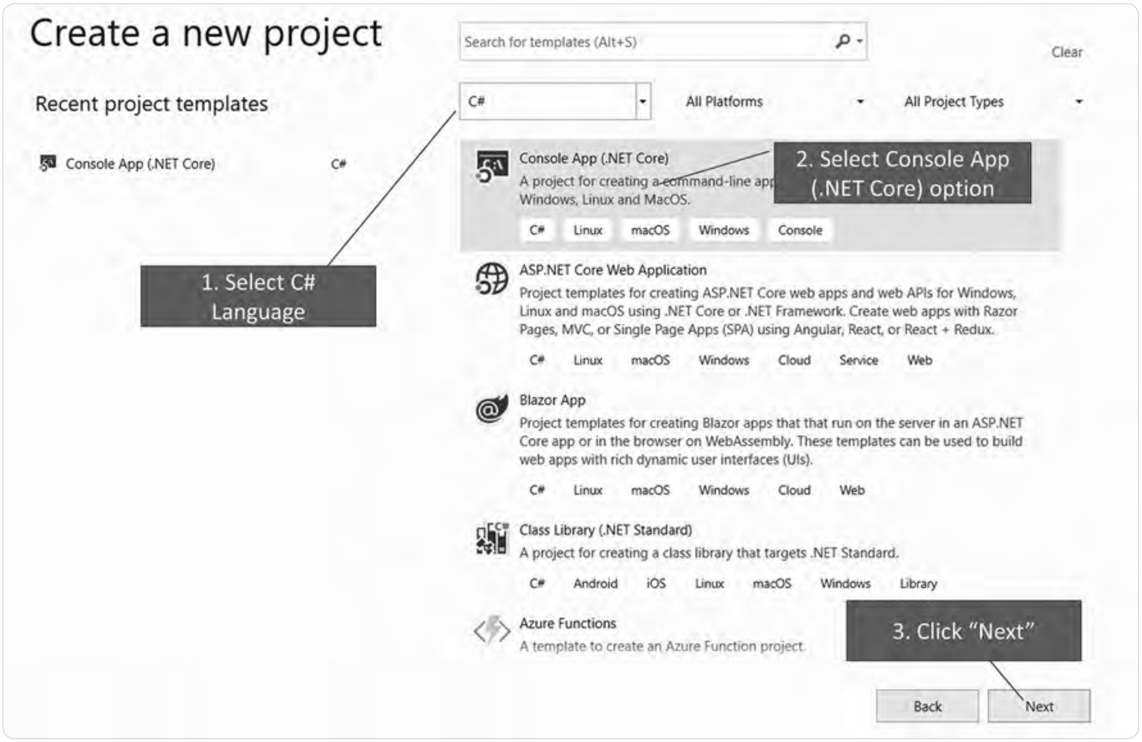
- Being Incremental and Iterative.

Step-By-Step Instructions

For this project example, you will need to launch a new project from the MVS file menu and create a Console App using the C# language (see Figure 3.1). A console application is one designed to be used for a text-only interface program. Although MVS can be used to create graphical user interfaces, games, mobile apps, and web-based applications, the console application is typically used for beginning programming due to its simple and straightforward characteristics.

Give your new project a name (the example in Figure 3.2 is named "MyConsole-App") and note the location where your project will be saved.

Once the program loads, you will see the boilerplate template text shown below in the large window on the left side of your screen. This is the main editor where your programming code is entered, and it is where you will spend most of your time. The smaller Solution Explorer window to the right shows all of the files that are part of a project, which lets you launch other files to open in the main editor.


Create a new project

Search for templates (Alt+S)

Clear

Recent project templates

Console App (.NET Core) C#

C#

All Platforms

All Project Types

Console App (.NET Core)
A project for creating a command-line ap
Windows, Linux and MacOS.
C# Linux macOS Windows Console

ASP.NET Core Web Application
Project templates for creating ASP.NET Core web apps and web APIs for Windows, Linux and macOS using .NET Core or .NET Framework. Create web apps with Razor Pages, MVC, or Single Page Apps (SPA) using Angular, React, or React + Redux.
C# Linux macOS Windows Cloud Service Web

Blazor App
Project templates for creating Blazor apps that that run on the server in an ASP.NET Core app or in the browser on WebAssembly. These templates can be used to build web apps with rich dynamic user interfaces (UIs).
C# Linux macOS Windows Cloud Web

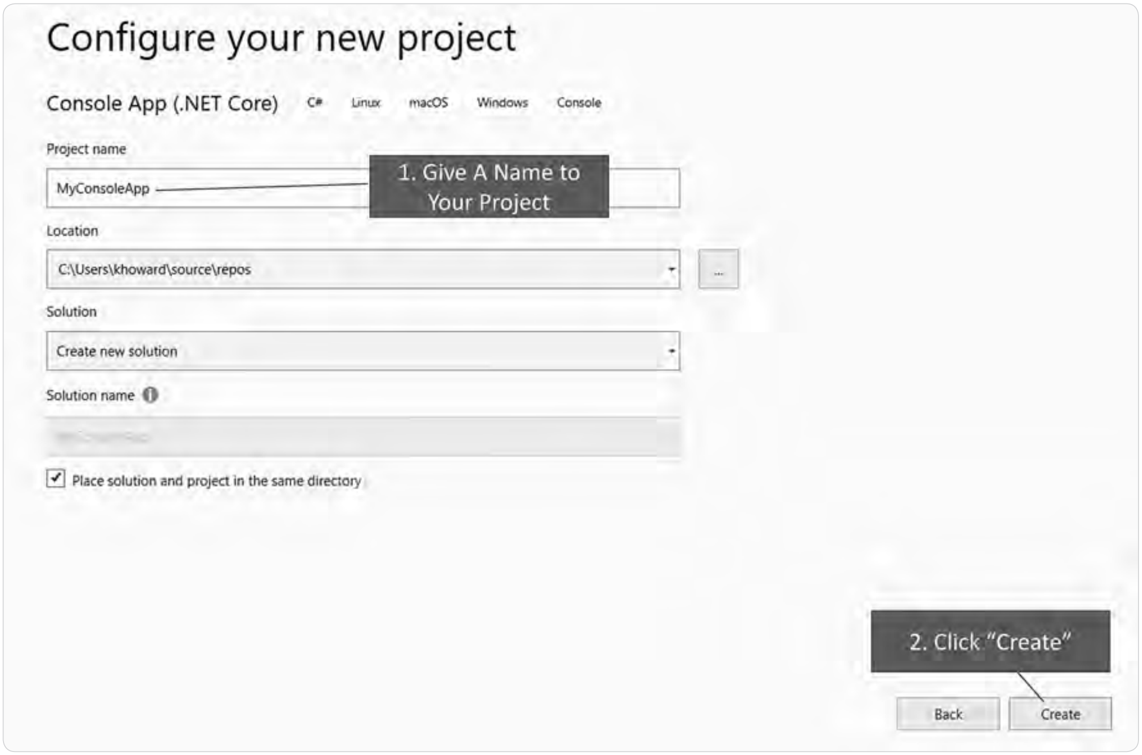
Class Library (.NET Standard)
A project for creating a class library that targets .NET Standard.
C# Android iOS Linux macOS Windows Library

Azure Functions
A template to create an Azure Function project.

Figure 3.1 Launch a new Visual Studio project.

The window along the bottom of the screen (see Figure 3.3) is the output window and will be used for debugging. For now, we will concentrate on the main editor window. Notice the use of curly braces ("{ }") to create a hierarchy in the code. Each beginning brace must have an ending brace, and the editing window depicts gray dotted lines to indicate which braces are paired. The rightmost set of braces contains the Main method, which tells the computer where your program begins. Your Main method created from the template should have the "Console.WriteLine("Hello World!");" line in it already. But if you have an older version of MVS, you may need to enter this line yourself. If you need to enter it, make sure to end the statement with ";". All statements in the Main method must end with a semicolon.

The Console.WriteLine statement is actually a separate method that executes specific code on the information contained in the parentheses. In this case, it tells the computer to write the line of text in the quotation marks ("Hello World!") to the


Configure your new project

Console App (.NET Core) C# Linux macOS Windows Console

Project name: MyConsoleApp

Location: C:\Users\khoward\source\repos

Solution: Create new solution

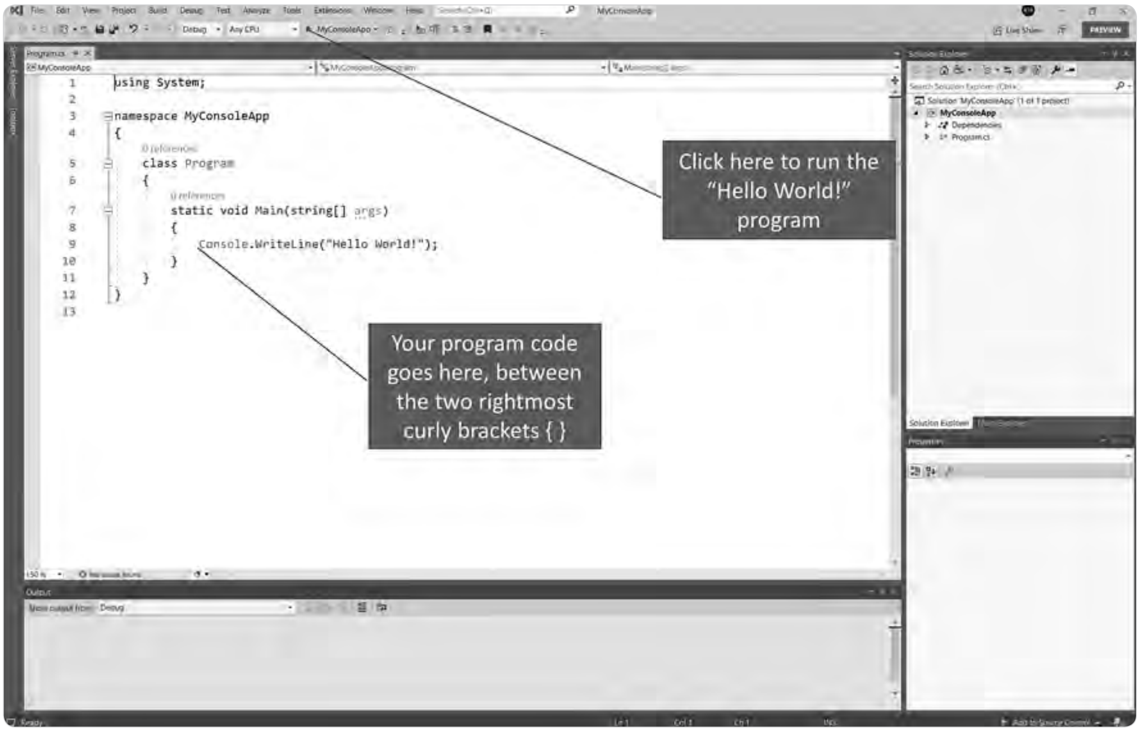
Solution name

✓ Place solution and project in the same directory

Figure 3.2 Name and create new project.

console application window. Although it's just a few lines of code, this represents a complete program that can be run across many types of computers to produce the same results that it does here. Have the students run the program by clicking on the green triangle on the icon bar above the main editor (see in Figure 3.3).

The output window displays the text contained in the Console.WriteLine statement (see Figure 3.4). While congratulations are in order for creating a program, the next step is to explore ways to add more functionality. According to Brennan and Resnick's Framework, this practice could be considered "Being Incremental and Iterative," whereby students apply an adaptive mindset as they test parts and play around with the code. Additionally, such an activity gives students an opportunity to strengthen their ability to "make sense of problems and persevere in solving them" (CCSS.Math. Practice.MP1).



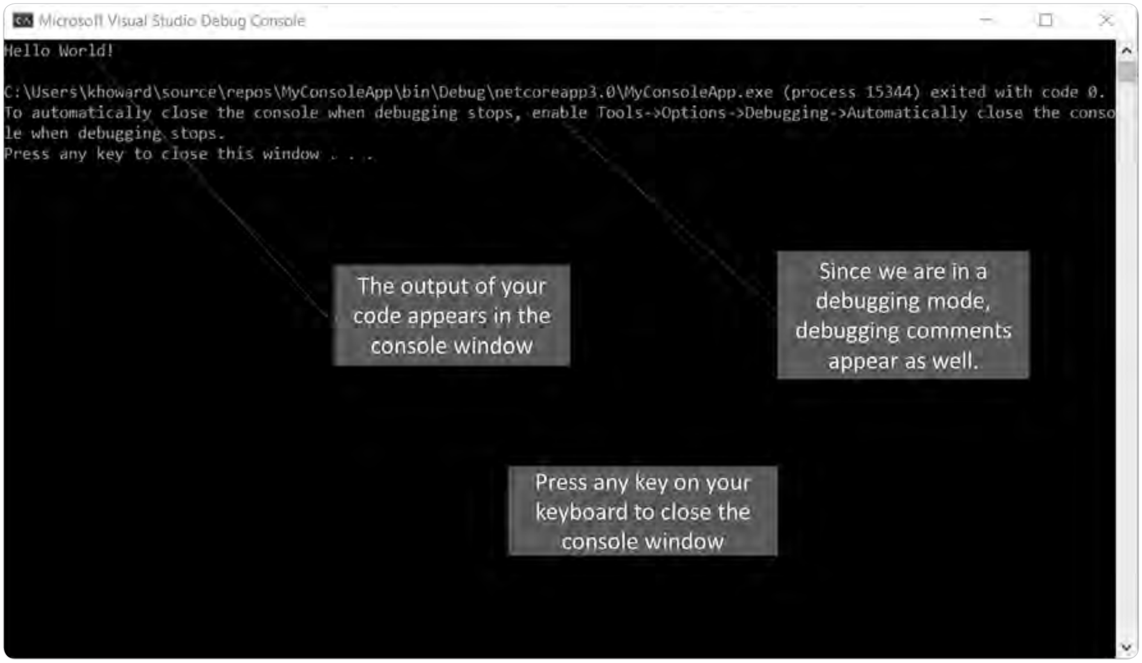
```
using System;

namespace MyConsoleApp
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

Click here to run the "Hello World!" program
Your program code goes here, between the two rightmost curly brackets { }

Figure 3.3 Project startup window containing *Hello World!* program boilerplate code.

One of the most important practices programmers will learn is commenting within their code. Comments help to narrate the action taking place in a program and provide insight into the thinking behind it. This will help the programmer, and others who read the code later, to understand what was intended when the code was written. Research has found that a programming learning approach requiring students to comment on their logic and intended structure before actually writing code improves the accuracy and proficiency of their coding (Sengupta, 2009). Unfortunately, due to the extra time it takes to provide thorough comments in code, even seasoned programmers don't always take the time to provide a solid roadmap for their programs.

The syntax used for writing comments in a programming environment varies across different languages, but in C# a line is rendered a comment when it starts with two forward slashes (//). This tells the computer that anything on the line after the

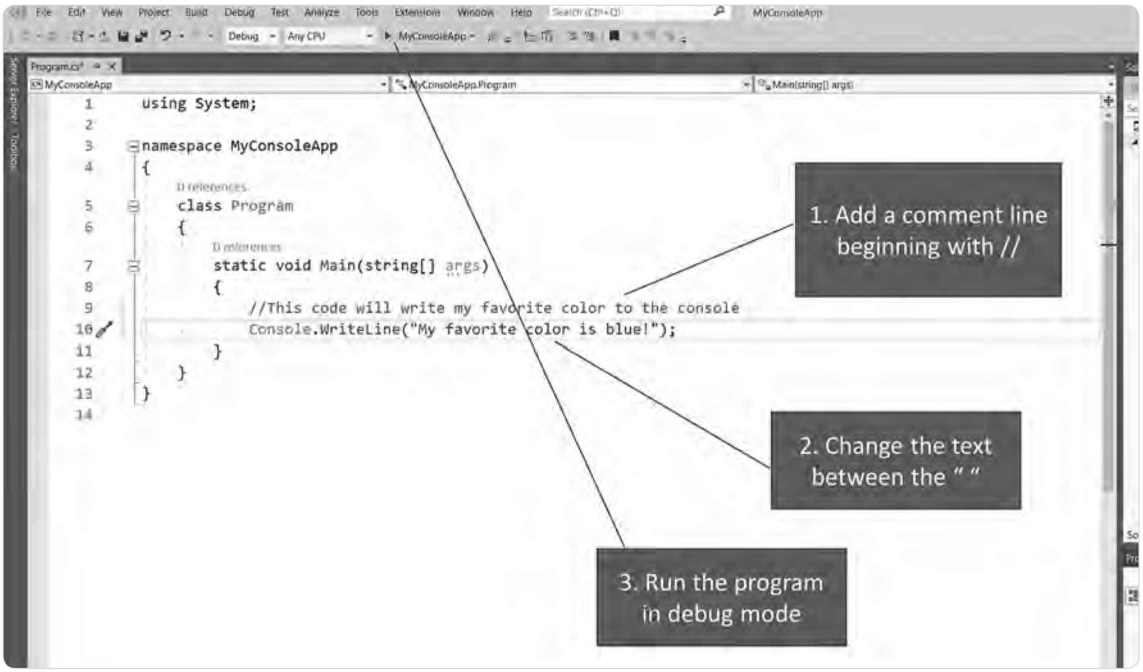


```
Hello World!

C:\Users\khoward\source\repos\MyConsoleApp\bin\Debug\netcoreapp3.0\MyConsoleApp.exe (process 15344) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```

Microsoft Visual Studio Debug Console
The output of your code appears in the console window
Since we are in a debugging mode, debugging comments appear as well.
Press any key on your keyboard to close the console window

Figure 3.4 Hello World! program output.

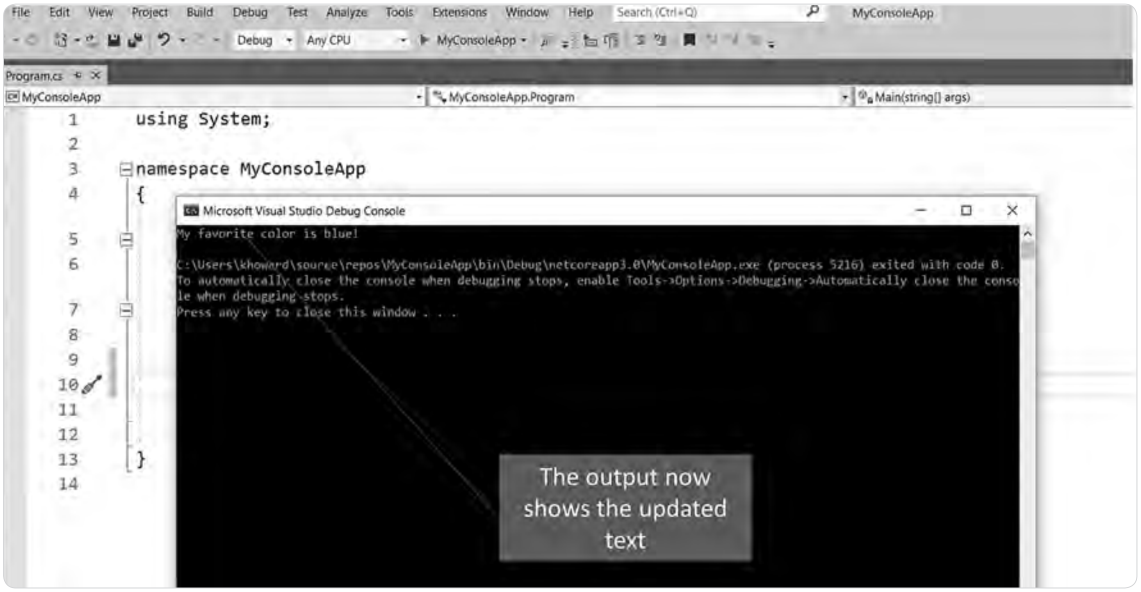
slashes is commentary and should not be executed as code. Text across multiple lines can be commented either with a beginning "/*" and an ending "*/" or by using double forward slashes at the beginning of each line. To cultivate good programming habits, beginners should be encouraged to document their steps as much as possible. For the exercise illustrated below, students would be instructed to write a comment line to describe the text that they will be printing to the console, followed by a WriteLine() method with the to-be-printed text in quotes. They can simply edit the existing WriteLine statement to reflect their own statement. Remind students to end the statement with a semicolon and to make sure they don't remove any of the curly braces; otherwise, they will receive errors when they run the program. As before, they will click the run icon to start the program in a console window.



```
using System;

namespace MyConsoleApp
{
    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            //This code will write my favorite color to the console
            Console.WriteLine("My favorite color is blue!");
        }
    }
}
```

File Edit View Project Build Debug Test Analyze Tools Extensions Window Help
Debug
Any CPU
MyConsoleApp
Program.cs
MyConsoleApp.Program
Main(string[] args)
1. Add a comment line beginning with //
2. Change the text between the " "
3. Run the program in debug mode

Figure 3.5 Add code comment and edit output code.



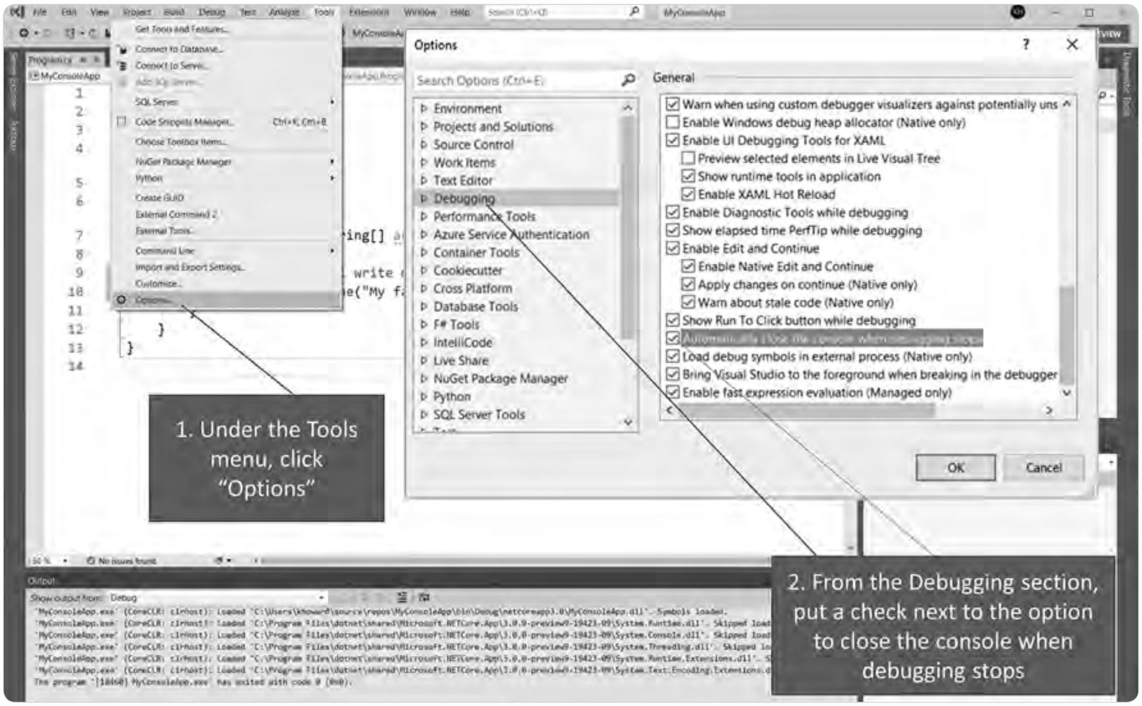
```
My favorite color is blue!

C:\Users\khoward\source\repos\MyConsoleApp\bin\Debug\netcoreapp3.0\MyConsoleApp.exe (process 5216) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```

Microsoft Visual Studio Debug Console
The output now shows the updated text

Figure 3.6 Program output after revisions.

The console app window now displays the students' revised text. As before, the console window can be closed by pressing any key on the keyboard. Next, we will change our options to clean up the debugging code that appears when we run our console application.

Under the Tools menu, select "Options" and then put a check next to the option shown in the graphic featured in Figure 3.7. Note that you may have to scroll to the bottom of the dialog box window to find the option. Click "OK" once you have checked the option.

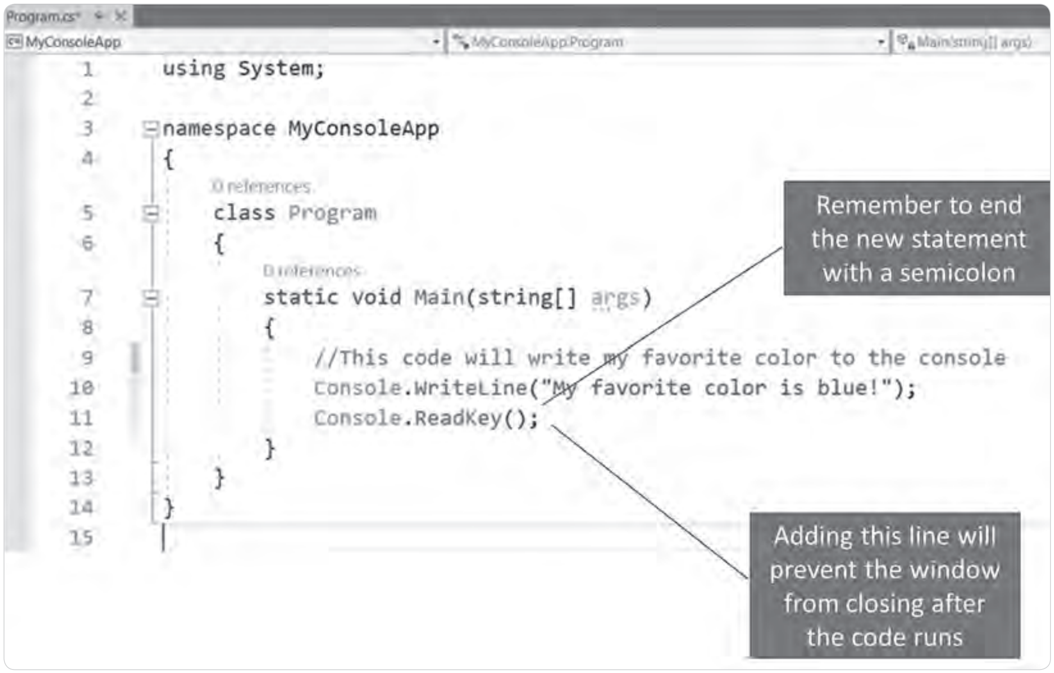


- [x] Warn when using custom debugger visualizers against potentially uns
- [ ] Enable Windows debug heap allocator (Native only)
- [x] Enable UI Debugging Tools for XAML
  - [ ] Preview selected elements in Live Visual Tree
  - [x] Show runtime tools in application
  - [x] Enable XAML Hot Reload
- [x] Enable Diagnostic Tools while debugging
- [x] Show elapsed time PerfTip while debugging
- [x] Enable Edit and Continue
  - [x] Enable Native Edit and Continue
  - [x] Apply changes on continue (Native only)
  - [x] Warn about stale code (Native only)
- [x] Show Run To Click button while debugging
- [x] Automatically close the console when debugging stops
- [x] Load debug symbols in external process (Native only)
- [x] Bring Visual Studio to the foreground when breaking in the debugger
- [x] Enable fast expression evaluation (Managed only)

```
Show output from: Debug
'MyConsoleApp.exe' (CoreCLR: clrhost): Loaded 'C:\Users\khoward\source\repos\MyConsoleApp\bin\Debug\netcoreapp3.0\MyConsoleApp.dll'. Symbols loaded.
'MyConsoleApp.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\3.0.0-preview9-19423-09\System.Runtime.dll'. Skipped load
'MyConsoleApp.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\3.0.0-preview9-19423-09\System.Console.dll'. Skipped load
'MyConsoleApp.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\3.0.0-preview9-19423-09\System.Threading.dll'. Skipped lo
'MyConsoleApp.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\3.0.0-preview9-19423-09\System.Runtime.Extensions.dll'. S
'MyConsoleApp.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\3.0.0-preview9-19423-09\System.Text.Encoding.Extensions.d
The program '[18460] MyConsoleApp.exe' has exited with code 0 (0x0).
```

Figure 3.7 Changing project options.

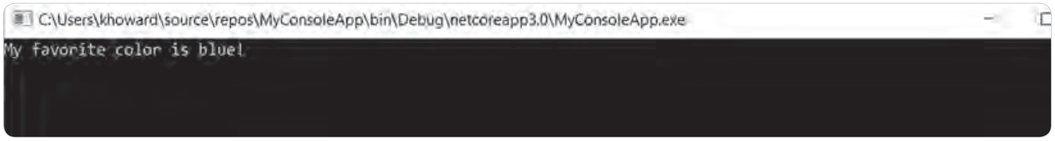
If you run the program now, you will no longer see the debugging text. However, it may run and close so quickly that you will not have a chance to read your output. Therefore, you'll want to add one additional statement to your program (shown in Figure 3.8).


Program.cs*
MyConsoleApp
MyConsoleApp.Program
Main(string[] args)
Remember to end the new statement with a semicolon
Adding this line will prevent the window from closing after the code runs

```
using System;

namespace MyConsoleApp
{
    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            //This code will write my favorite color to the console
            Console.WriteLine("My favorite color is blue!");
            Console.ReadKey();
        }
    }
}
```

Figure 3.8 Adding *Console.ReadKey*(); statement.

The additional statement will allow your program to run without the extra debugging text, and your output will be clean, containing only the information you programmed it to write (see Figure 3.9).

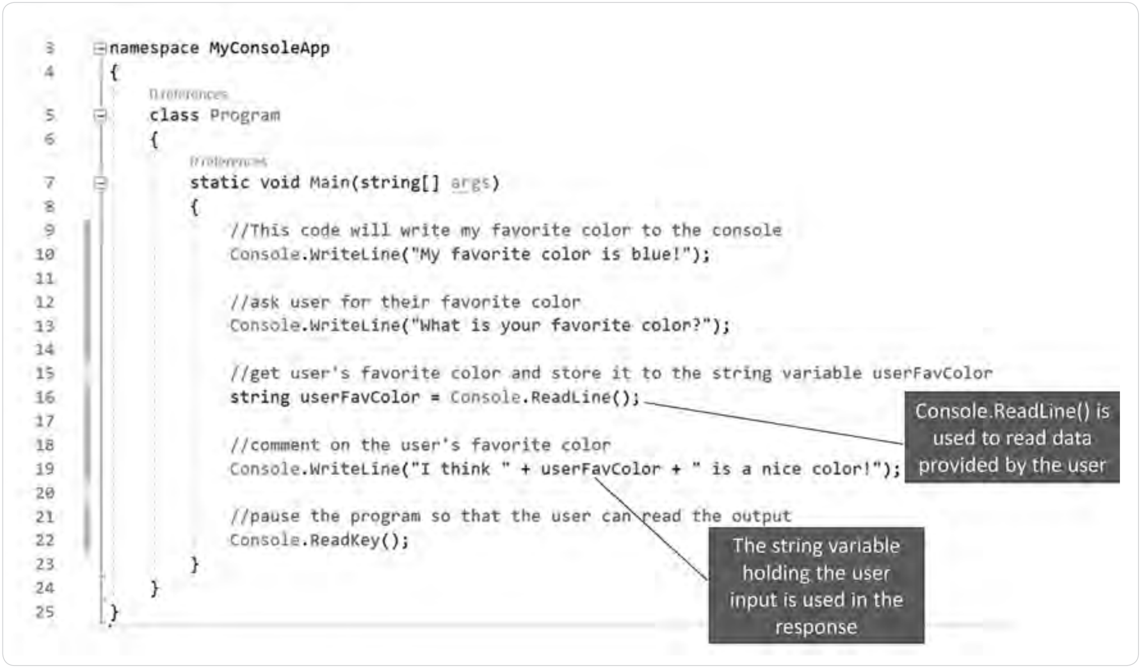

C:\Users\khoward\source\repos\MyConsoleApp\bin\Debug\netcoreapp3.0\MyConsoleApp.exe

```
My favorite color is blue!
```

Figure 3.9 Program output after project option change.

Now let's move on to working with *variables*. We'll start out with the string type variable, which is used frequently in programming. As previously noted, string variables are used to hold data that will be identified and manipulated as text. Although a text variable can store numbers, it stores them as text as well. This means that you cannot perform calculations on any numbers stored in them (without first converting them), just as you cannot perform meaningful calculations on words or letters. To perform calculations, the data needs to be stored in a numeric variable such as an integer or float variable type. The next exercise will allow us to manipulate data stored in string variables and will highlight the kinds of issues that can arise when they are treated as numeric variables in most programming languages.

Have students update their console application code to read as shown in Figure 3.10. The code includes a comment for each line of text to explain what the programmer intends to happen. Experienced programmers usually do not comment on



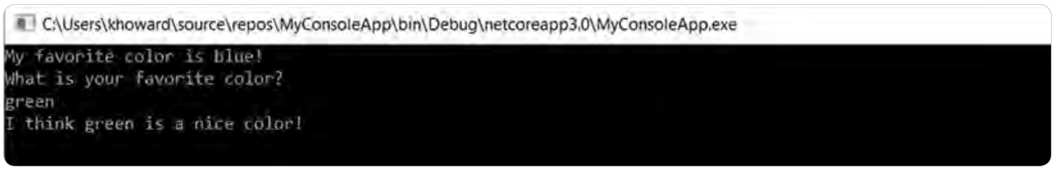
```
namespace MyConsoleApp
{
    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            //This code will write my favorite color to the console
            Console.WriteLine("My favorite color is blue!");

            //ask user for their favorite color
            Console.WriteLine("What is your favorite color?");

            //get user's favorite color and store it to the string variable userFavColor
            string userFavColor = Console.ReadLine();

            //comment on the user's favorite color
            Console.WriteLine("I think " + userFavColor + " is a nice color!");

            //pause the program so that the user can read the output
            Console.ReadKey();
        }
    }
}
```

Console.ReadLine() is used to read data provided by the user
The string variable holding the user input is used in the response

Figure 3.10 Completed project code.

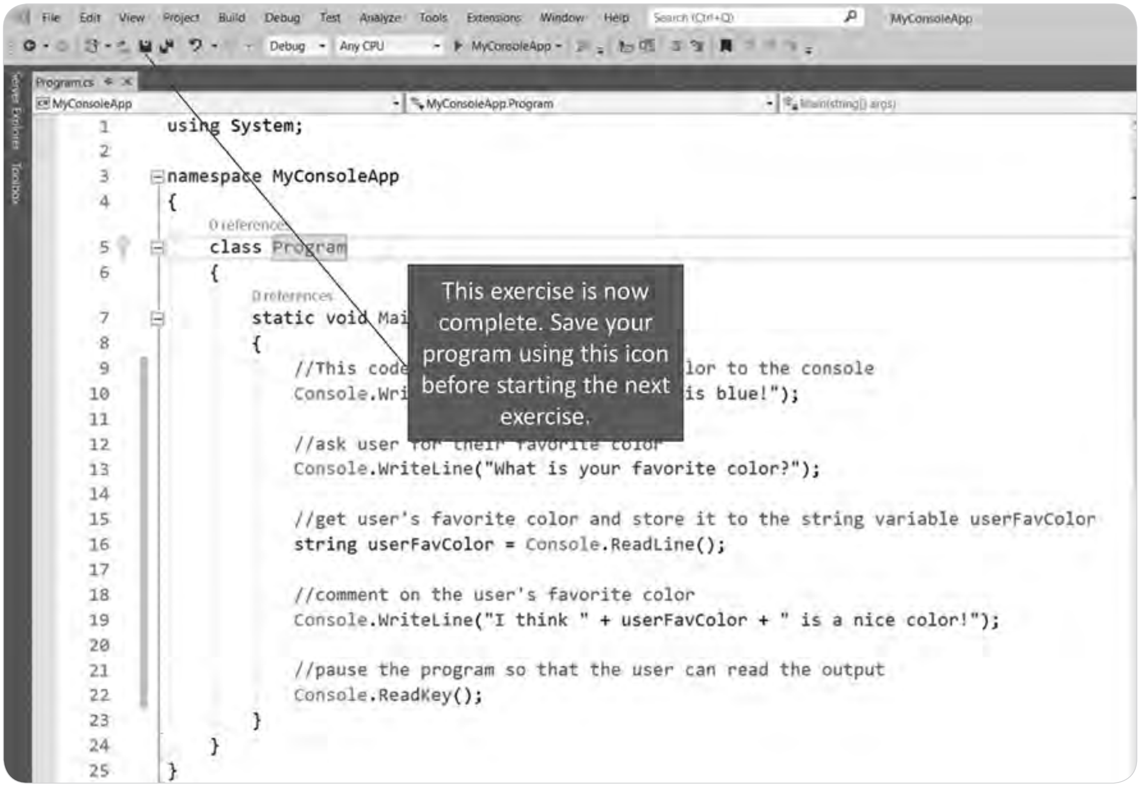
each line of code; instead they comment on selected lines or entire blocks of code at a time. However, beginners can benefit from having to think through and explain their programming intentions and will find these explanations helpful when trying to pick up on a project days later. The code on line 10 has not changed (note that your line numbers may be different depending on your spacing). Line 13 asks the user what their favorite color is and stores their answer in a string variable named "userFavColor". A new command is introduced on line 16: Console.ReadLine(), which reads the user's response to the question and assigns it to the string userFavColor. Note that the string userFavColor is being declared and assigned the value from Console.ReadLine() in the same line.

Line 19 uses the data provided by the user (favorite color) as part of the sentence responding to the user. The user's favorite color is stored in the string variable userFavColor and is added to sentence text before and after it. Note the use of "+" signs when adding strings and/or string variables together. Also note that spaces need to be entered in the quote marks to keep the words from all running together when the strings are combined. A comment has been added above the Console.ReadKey() statement to explain its purpose. The output of the program is shown in Figure 3.11.



```
C:\Users\khoward\source\repos\MyConsoleApp\bin\Debug\netcoreapp3.0\MyConsoleApp.exe
My favorite color is blue!
What is your favorite color?
green
I think green is a nice color!
```

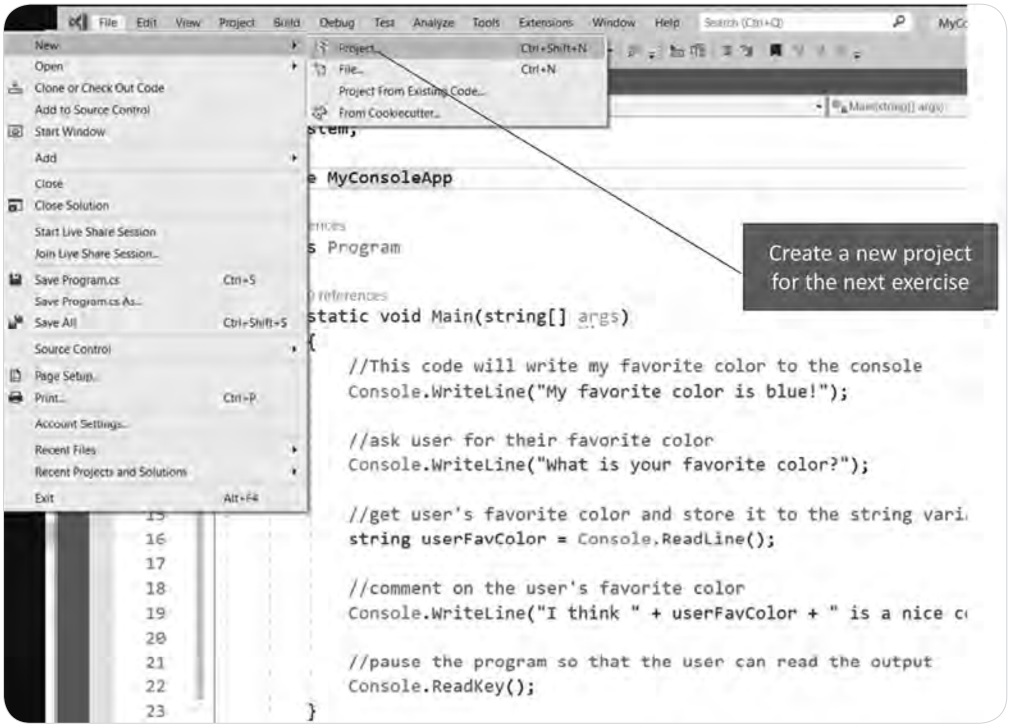
Figure 3.11 Program output from completed project.


File Edit View Project Build Debug Test Analyze Tools Extensions Window Help
Search (Ctrl+Q)
MyConsoleApp
Debug
Any CPU
MyConsoleApp
Program.cs
MyConsoleApp
MyConsoleApp.Program
Main(string[] args)
This exercise is now complete. Save your program using this icon before starting the next exercise.

```
using System;

namespace MyConsoleApp
{
    0 references
    class Program
    {
        0 references
        static void Mai
        {
            //This code                  lor to the console
            Console.Wri                  is blue!");

            //ask user for their favorite color
            Console.WriteLine("What is your favorite color?");

            //get user's favorite color and store it to the string variable userFavColor
            string userFavColor = Console.ReadLine();

            //comment on the user's favorite color
            Console.WriteLine("I think " + userFavColor + " is a nice color!");

            //pause the program so that the user can read the output
            Console.ReadKey();
        }
    }
}
```

Figure 3.12 Saving the project.

Before moving on to the next exercise, save your program file as shown in Figure 3.12.



```
stem;

e MyConsoleApp

s Program

static void Main(string[] args)
{
    //This code will write my favorite color to the console
    Console.WriteLine("My favorite color is blue!");

    //ask user for their favorite color
    Console.WriteLine("What is your favorite color?");

    //get user's favorite color and store it to the string vari
    string userFavColor = Console.ReadLine();

    //comment on the user's favorite color
    Console.WriteLine("I think " + userFavColor + " is a nice c

    //pause the program so that the user can read the output
    Console.ReadKey();
}
```

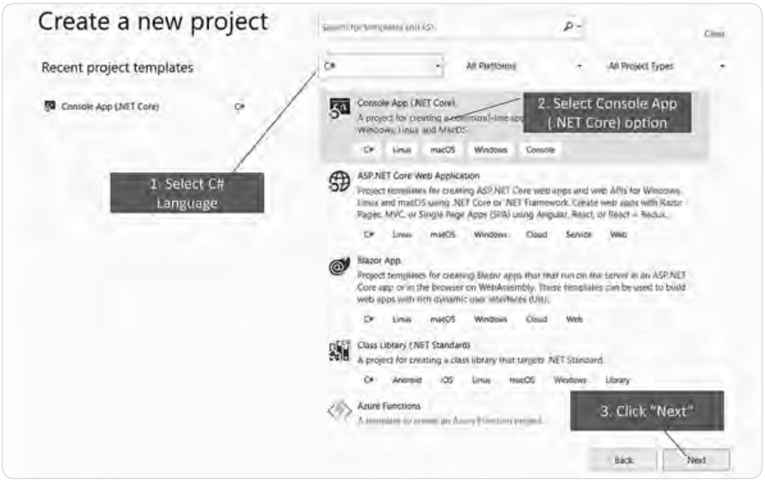
Figure 3.13 Start new project.

For the final exercise in this chapter, students explore declaring mathematical and Boolean variables. We'll start off with a new project using the menu option shown in Figure 3.13.

From the dialog window, select the C# language option and Console App, as before (see Figure 3.14).

Name the project "MyVariableApp" and create the project (see Figure 3.15).

The Console.WriteLine("Hello World!") line can be replaced with the MyVariable-App code (downloaded from the book website) to save time. However, if an instructor would like to provide students with practice coding from scratch and working in the MVS code editor, they can enter the code shown in Figure 3.16 manually.



Create a new project

Recent project templates

Console App (.NET Core) C#

C# | All Platforms | All Project Types

Console App (.NET Core)
A project for creating a command-line application that can run on .NET Core on Windows, Linux and MacOS.
C# Linux macOS Windows Console

ASP.NET Core Web Application
Project templates for creating ASP.NET Core web apps and web APIs for Windows, Linux and macOS using .NET Core or .NET Framework. Create web apps with Razor Pages, MVC, or Single Page Apps (SPA) using Angular, React, or React + Redux.
C# Linux macOS Windows Cloud Service Web

Blazor App
Project templates for creating Blazor apps that that run on the server in an ASP.NET Core app or in the browser on WebAssembly. These templates can be used to build web apps with rich dynamic user interfaces (UIs).
C# Linux macOS Windows Cloud Web

Class Library (.NET Standard)
A project for creating a class library that targets .NET Standard.
C# Android iOS Linux macOS Windows Library

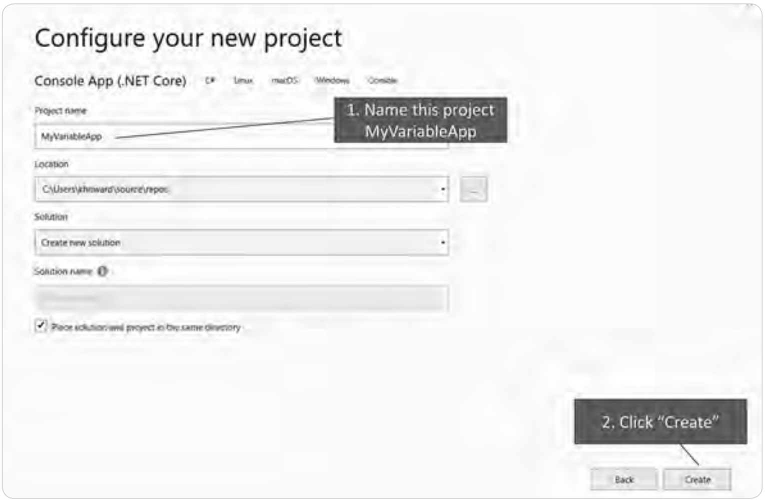
Azure Functions
A template to create an Azure Function project.

1. Select C# Language

2. Select Console App (.NET Core) option

3. Click "Next"

Figure 3.14
Select programming language and project type.



Configure your new project

Console App (.NET Core) C# Linux macOS Windows Console

Project name: MyVariableApp

Location: C:\Users\khoward\source\repos\

Solution: Create new solution

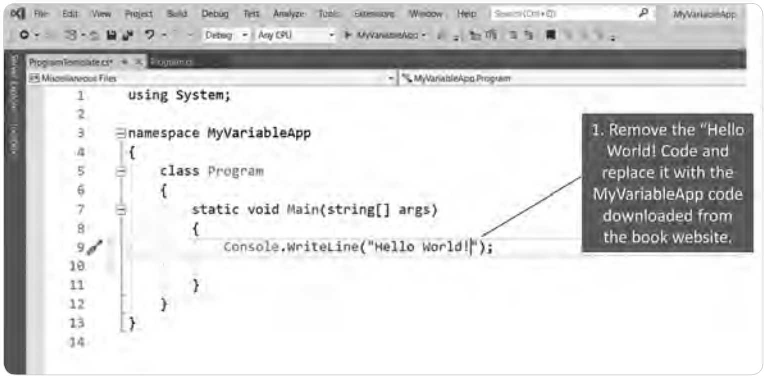
Solution name

☑ Place solution and project in the same directory

1. Name this project MyVariableApp

2. Click "Create"

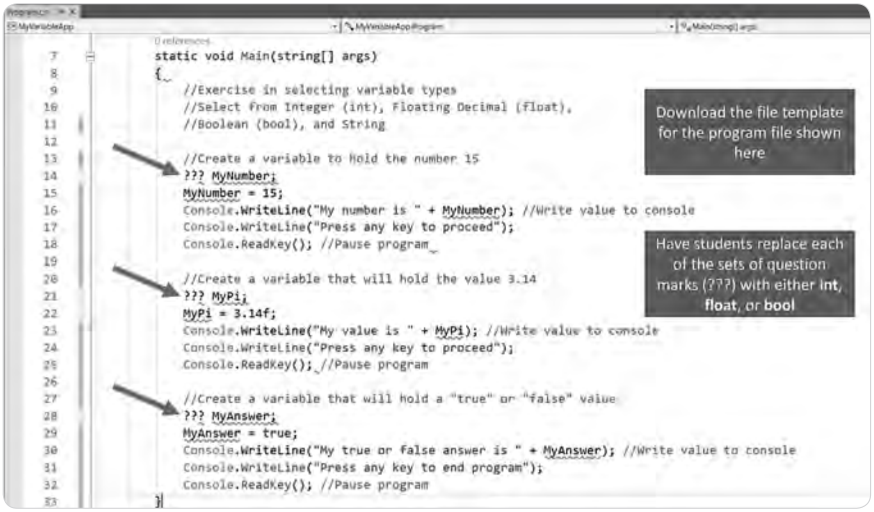
Figure 3.15
Name new project "MyVariableApp".



```
using System;

namespace MyVariableApp
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");

        }
    }
}
```

1. Remove the "Hello World! Code and replace it with the MyVariableApp code downloaded from the book website.

Figure 3.16
Replace Console. WriteLine("Hello World!") statement with project code shown in next figure (3.17).



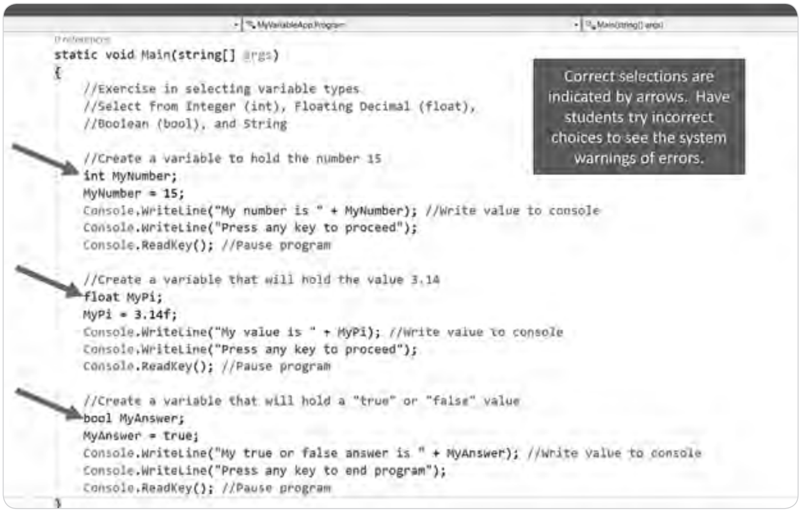
```
static void Main(string[] args)
{
    //Exercise in selecting variable types
    //Select from Integer (int), Floating Decimal (float),
    //Boolean (bool), and String

    //Create a variable to hold the number 15
    ??? MyNumber;
    MyNumber = 15;
    Console.WriteLine("My number is " + MyNumber); //Write value to console
    Console.WriteLine("Press any key to proceed");
    Console.ReadKey(); //Pause program

    //Create a variable that will hold the value 3.14
    ??? MyPi;
    MyPi = 3.14f;
    Console.WriteLine("My value is " + MyPi); //Write value to console
    Console.WriteLine("Press any key to proceed");
    Console.ReadKey(); //Pause program

    //Create a variable that will hold a "true" or "false" value
    ??? MyAnswer;
    MyAnswer = true;
    Console.WriteLine("My true or false answer is " + MyAnswer); //Write value to console
    Console.WriteLine("Press any key to end program");
    Console.ReadKey(); //Pause program
}
```

Download the file template for the program file shown here

Have students replace each of the sets of question marks (???) with either **int**, **float**, or **bool**

Figure 3.17
Type or download the code shown here from k12stemequity.com. Replace "???" with "int", "float", or "bool".

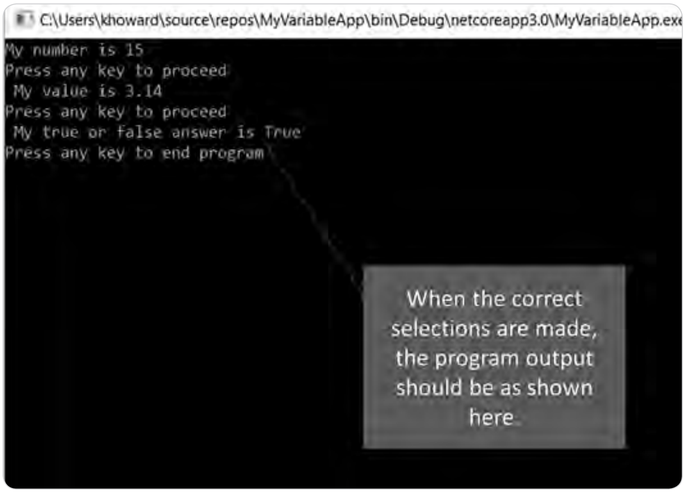
Figure 3.17 shows the program after the "Hello World!" statement has been replaced by the template text for this exercise. The code shown only needs to have the three variable declarations completed with the variable type required for the respective data descriptions. The arrows point to placeholder text (???) that needs to be replaced with either "int", "float", or "bool" based on the data requirements for each. You will notice wavy red lines below some of the text in the program similar to the spell-check functionality seen in many word processors and browsers. In this case, the warnings are part of an extensive code-checking function Microsoft calls



```
static void Main(string[] args)
{
    //Exercise in selecting variable types
    //Select from Integer (int), Floating Decimal (float),
    //Boolean (bool), and String

    //Create a variable to hold the number 15
    int MyNumber;
    MyNumber = 15;
    Console.WriteLine("My number is " + MyNumber); //Write value to console
    Console.WriteLine("Press any key to proceed");
    Console.ReadKey(); //Pause program

    //Create a variable that will hold the value 3.14
    float MyPi;
    MyPi = 3.14f;
    Console.WriteLine("My value is " + MyPi); //Write value to console
    Console.WriteLine("Press any key to proceed");
    Console.ReadKey(); //Pause program

    //Create a variable that will hold a "true" or "false" value
    bool MyAnswer;
    MyAnswer = true;
    Console.WriteLine("My true or false answer is " + MyAnswer); //Write value to console
    Console.WriteLine("Press any key to end program");
    Console.ReadKey(); //Pause program
}
```

Correct selections are indicated by arrows. Have students try incorrect choices to see the system warnings of errors.

Figure 3.18
Project code with the correct variable types entered.



```
C:\Users\khoward\source\repos\MyVariableApp\bin\Debug\netcoreapp3.0\MyVariableApp.exe
My number is 15
Press any key to proceed
 My value is 3.14
Press any key to proceed
 My true or false answer is True
Press any key to end program
```

When the correct selections are made, the program output should be as shown here.

Figure 3.19 Corrected program output.

"Intellisense," which alerts the programmer that some syntax needs to be corrected. Just as with word processors, the user can hover over the wavy red line to activate popup boxes that provide clues as to what needs to be corrected. The use of programming environments with functions such as these assists beginners in learning syntax and finding common errors before they get to the debugging stage.

In this case, the variables have warnings on them because they have not been properly declared in the program. Once students have been made aware of the different kinds of data that can be stored in each data type, selecting the correct answers should be pretty straightforward. However, it is helpful to encourage them to attempt to select incorrect data types to learn how the editor guides them to correct their mistakes.

Figure 3.18 displays the program completed with the correct variable types entered in place of the placeholders. Notice that once the correct answers were entered, all of the wavy red lines disappeared and the program runs without issues.

Figure 3.19 shows the output to the console window when the program is run. In each case, the program stored and wrote each of the variable values as expected.

Mathematical Connections

Although the program compiled to complete the console application project is a rather simple one in terms of functionality, it engages a number of math concepts that strengthens students' understanding of them. The discussion of variable types starts to form their understanding of the differences between integers and floating decimals, even if they are not yet ready to perform calculations. The assignment of data to a variable (albeit text data) starts to plant the idea of using a variable to represent something else, as well as using that variable as part of a larger process. Learning variable types should include discussing how the Integer type can be used to hold whole numbers. Common Core State Standards for Mathematics (CCSSM) call for students to be performing addition and subtraction operations with positive integers as early as kindergarten (K.CC, K.OA) and multiplication and division of positive integers by grade 3 (3.OA, 3.NBT). By grade 4 , they call for students to understand decimal notation for fractions (4.NF) and to perform operations with decimals to hundredths by grade 5 (5.NBT), both of which align with the Float variable type. Students are expected to determine whether equations involving addition and subtraction are true or false by grade 1 (1.OA), giving context to the Boolean data type. Given that these are concepts that students are expected to start grasping in K–5 classrooms, programming approaches that insulate students from the thought processes necessary to declare variables in code miss an opportunity to help shore up their foundation in some basic math and CS concepts.

Dr. Cory Gleasman shares his approach to preparing preservice teachers to use coding with their math instruction. He challenges the notion of putting too much focus on a specific tool or platform, warning of the potential developmental harm that can come from bringing the math content in as an afterthought to a coding lesson, rather than purposefully integrating it within. His insights inform elementary teacher education in CS and math pedagogy.

Coding + Math during Preservice Teacher Development

DR. CORY GLEASMAN
Assistant Professor of Computer Science Education at Tennessee Technological University

I've witnessed the lack of K–5 CS education provided by university teacher preparation programs. Not only are K–5 preservice teachers apprehensive of the terms *computer science* and *coding*, but so are teacher educators who are preparing our future teachers. Computer science and coding are not the same and are frequently mistaken for one another, especially by preservice teacher candidates. Computer science consists of broad overarching theories of computing. While coding is a skillful act contingent upon the understanding of CT concepts and computing knowledge bases, when instructing elementary preservice teachers I have found explaining this differentiation to be imperative for them to be successful with instruction on lesson plans involving block-based coding. For the past four years, I have been researching the cross sections of K–5 mathematics and block-based coding languages in order to offer preservice teachers a practical approach to integrating CS into their instruction. Furthermore, I have set out to establish a set of design guidelines for teacher educators to follow if they wish to create a higher-education learning environment, enabling K–5 preservice teachers to integrate coding into their mathematics instruction. I have placed an emphasis on viewing the cross section between K–5 mathematics and coding through a computational thinking lens and then using the intersection of computational thinking to create math teaching opportunities.

Through multiple instructional and research study iterations, I developed a five-week intervention course called Block-Based Coding and Computational Thinking for Conceptual Mathematics (B2C3Math). It was created to serve as a template for integrating block-based coding into elementary preservice teacher mathematics methods courses. The feedback surrounding the implementation of this intervention course has been extremely positive. The B2C3Math intervention was implemented three times at the University of Georgia and is currently in its fourth iteration at Tennessee Technological University.

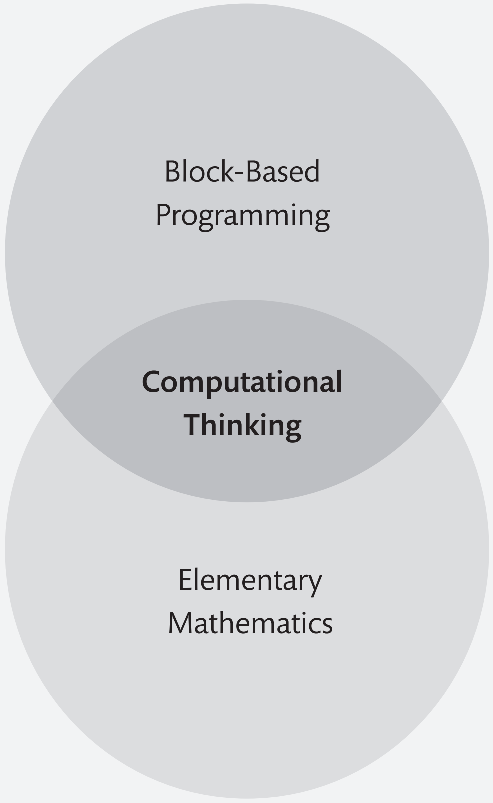


Figure 3.20 Cross section between mathematics and coding (photo credit: Gleasman & Kim, 2018).

The notion of using coding as a tool to facilitate mathematics is not new. In the 1980s, Seymour Papert detailed his coding-mathematics learning theories in his memorable book *Mindstorms: Children, Computers, and Powerful Ideas*, and they are still relevant today. What is different is the plethora of tools and accompanying frameworks at our disposal. Many times it becomes overwhelming having so many coding platforms as options, and a focus is placed on the tool and the products made from manipulating the tool. Within B2C3Math, preservice teachers are taught to ensure learning opportunities are occurring during the coding process and not dependent upon a coded product. I preach that coding platforms can and will forever change; however, the underlying concepts can be engaged to elicit similar mathematics learning regardless of the technologies in play. The focus on the overlapping of math and coding concepts in the form of CT helps promote the learning of both math and coding, not just the procedural learning of a block-based coding language. I witness preservice teachers rely on drag-and-drop coding curriculums, which are time-efficient, but can be more harmful than good, especially at the developmental level. Helping preservice teachers understand how to develop a coding-mathematics activity has been more beneficial than teaching a mapped-out coding curriculum during teacher preparation. During the first B2C3Math iteration, preservice teachers tended to plan a lesson where their future students were coding; however, math learning followed the coding process and was dependent upon visual outputs. Design guidelines and instruction associated with

B2C3Math were altered to ensure preservice teachers evoked math learning to occur simultaneously with coding processes.

Much data and relevant findings have surfaced as a result of research surrounding B2C3Math, but the most impactful and eye-opening conclusion I realize after its implementation is preservice teachers enjoy the innovation and creativity required to create coding-math lesson plans. They have expressed that coding and CT are an avenue in which to teach elementary math. While participating in the B2C3Math course, many preservice teachers were hesitant to develop coding lesson plans; however, I have witnessed preservice teachers alter their attitudes and perceptions. They now view the incorporation of block-based coding-math lessons as a feasible way to enhance their future students' mathematical understanding.

For more information regarding the B2C3Math intervention, see our article in Springer's *Digital Experiences in Mathematics Education Journal* titled "Pre-Service Teachers' Use of Block-Based Programming and Computational Thinking to Teach Elementary Mathematics" (Gleasman & Kim, 2020).

Mission Clarity

Working with data is a fundamental expectation at the core of computer programming. Number sense, as the foundation upon which mathematics is built, is foundational to students' deep understanding of all branches of mathematics. Likewise, data sense is at the heart of programming. Understanding how to manipulate different forms of data is crucial to becoming a good programmer. Understanding how to declare, initiate, manipulate, read, write, and store data takes purposeful effort.

If programming in K–5 education is viewed as primarily a means of self-expression with just a *side order* of CT, then the effort it takes to understand the fundamentals of programming may seem too steep to be entirely necessary. Conversely, if the long view of preparing students for secondary education, and ultimately for potential careers in information technology, is taken, then it is imperative that we familiarize students with the most basic of concepts that comprise CS. Whichever purpose an educator pursues in this regard, we encourage them to make students and parents partners in the decision-making process. Not every student aspires to have a career as a computer programmer one day, nor is every student longing for tinkering and self-expression through digital channels. Thus, a one-size-fits-all model of CS is not likely to best serve all students. Building a CS program around just one approach is a surefire way to ensure that some students' wants and needs will be excluded.

RESOURCES

RESOURCE	SOURCE	LINK
C# Data Types Reference	Tutorials Point	(iste.org/standards/ computational-thinking)
Java Data Types Reference	Tutorials Point	(tutorialspoint.com/ Data-types-in-Java)
C# Reference for Built-in Data Types	Microsoft Corporation	(docs.microsoft.com/en-us/ dotnet/csharp/language-reference/ keywords/built-in-types-table)
Visual Studio Download	Microsoft Corporation	(visualstudio.microsoft.com/ downloads)