**Topic 1: Methods**

Top of Form

Bottom of Form

**Content**

**Reading**

[Reading](https://dmacc.blackboard.com/webapps/blackboard/content/listContent.jsp?course_id=_102593_1&content_id=_7286947_1)

Methods

* + Microsoft:docs.microsoft.com
    - [Methods](https://docs.microsoft.com/en-us/dotnet/csharp/methods)
  + GeeksForGeeks: geeksforgeeks.com
    - [Methods](https://www.geeksforgeeks.org/c-sharp-methods/)
  + TutorialsPoint: tutorialspoint.com
    - [Methods](https://www.tutorialspoint.com/csharp/csharp_methods.htm)
  + W3Schools: w3schools.com
    - [Methods](https://www.w3schools.com/cs/cs_methods.asp)
    - [Method Parameters](https://www.w3schools.com/cs/cs_method_parameters.asp)

**Methods**

[Methods](https://dmacc.blackboard.com/webapps/blackboard/content/listContent.jsp?course_id=_102593_1&content_id=_7286947_1)

Methods break down code into smaller, easier to maintain modules. A method with no return type executes statements and ends. A value-returning method executes statements and returns a value to the method call. A method call is the statement that invokes, or initiates the method code at that point in the program.

It is nice to have method called getUserInputInt() in your Main method for several reasons.

* + Readability-your Main method is smaller, easier to read (and thus maintain)
  + Robustness and Elegance-you can maintain error and exception handling in the method
  + Modularity-you can reuse the method getUserInputInt() more easily that copy paste in the same program and in other programs

You have already used methods. Here is an an example of a method call:

public static void Main(string[] args)

{

String fullName = "Ayah Morgana";

String middleName = fullName.Substring(5); // Call to method Substring(int) from Class String

Console.WriteLine(middleName);

}

You have already used other methods from the String class, here are the signatures of a few methods

* + String Substring(int index, int length) // This is an overloaded method (more on that later!)
  + bool Equals(String)
  + int CompareTo(String)

**Why are methods useful? What are they anyway?**

[Why are methods useful? What are they anyway?](https://dmacc.blackboard.com/webapps/blackboard/content/listContent.jsp?course_id=_102593_1&content_id=_7286947_1)

Methods are a way to break code into subroutines. If you are planning to call the same line or block of code multiple times, it is useful to have the code in a subroutine (in C# method.) It can also make your code more readable. Instead of interpreting calculations or analyzing several lines of code, a method name has

he clear advantage of telling the programming what is happening: called CalculateArea, CalculateTaxOwed, DisplayCurrency, UpdateName, ...

This means if you are repeating the same code or if you want to organize your code, you can make a method and call the method. You will need to pass the method what it needs to complete its task.

For a simple example, if you wish to cacluate tax, if it is all in the Main method (yes, you have already been using a method! ) then is looks like the following:

class Program

{

public static void Main()

{

const decimal TAX\_RATE = 0.07m;

// Calculate book tax

decimal bookPrice = 25.99m;

decimal bookTax;

bookTax = bookPrice \* TAX\_RATE:

Console.WriteLine("Book tax is {0:C}", bookTax);

// Calculate meal tax

decimal mealPrice = 12.39m;

decimal mealTax;

mealTax = mealPrice \* TAX\_RATE;

Console.WriteLine("Meal tax is {0:C}", mealTax);

}

}

Making a method, adds to ease of the code reading. While cacluting tax is a simple one calucation, methods can be more complex and accomplish several related tasks at once.

Below, you see the method CalcuateTax. Notice in main, the method call CacluateTax does not need you to interpret the calculation.

class Program

{

private static decimal CalculateTax(decimal price, decimal taxRate)

{

decimal tax;

tax = price \* taxRate;

return tax;

}

public static void Main()

{

const decimal TAX\_RATE = 0.07m;

// Calculate book tax

decimal bookPrice = 25.99m;

decimal bookTax;

bookTax = CalculateTax(bookPrice, TAX\_RATE);

Console.WriteLine("Book tax is {0:C}", bookTax);

// Calculate meal tax

decimal mealPrice = 12.39m;

decimal mealTax;

mealTax = CalculateTax(mealPrice, TAX\_RATE);

Console.WriteLine("Meal tax is {0:C}", mealTax);

}

}

This requires more explanation! What do you mean by "passing" what are all the parts? Keep Reading!

**Breaking down a Method**

[Breaking down a Method](https://dmacc.blackboard.com/webapps/blackboard/content/listContent.jsp?course_id=_102593_1&content_id=_7286947_1)

First, we break down the parts of the method, lets use CalculateTax as an example:

static private decimal CalculateTax(decimal price, decimal taxRate)

{

decimal tax;

tax = price \* taxRate;

return tax;

}

* + First, is the method signature
    - static private decimal CalculateTax(decimal price, decimal taxRate)
    - static is a keyword indicating it is not a class method (more on that later, for now just remember to include it)
    - private is the keyword that describes what code can access the method. Possibilities are private, protected, and public. Here private means only class Program can see and use this method. It could be public as well, meaning another class could access it as well.
    - decimal is the return type of the method. This means the method will return a value of type decimal. Methods can return double, int, any data type, or they can also return nothing. In the case of method that does not return a value, the keyword void is as the return type.
    - (decimal price, decimal taxRate) is the parameter list or argument list, encased in parenthesis (). This list passes in information needed in the method. You will need to pass information into the method as it will not have access to variables declared in any other method, even Main method. If your method is performing a task and does not need to pass information, you can have an empty parameter list, but you still need the parenthesis (), they are simply empty to indicate an empty parameter list
  + Inside the curly brackets {} is the method body, this is what
    - {
    - decimal tax;
    - tax = price \* taxRate;
    - return tax;
    - }
    - Inside you have local varibles. Here tax is a local variable. These variables are not visible or accessible to the Main method or other methods you may write. Once the code execution exits the methods, the local variables and there values are gone.
    - The caclulation is included, the code that performs the method task "CalculateTax". This need not be a calcuation. It could be DisplayCurrency, CalcuateArea, or any task needed in your program.
    - return is the final line of this method. If your method is of type void, you will not need this return. This return must return the value specified as the return type in the method signature. A method expecting a string to be return, must return a string, or you will have a compile error. Here the return type is decimal, the local variable declared and used for calculation is also declared to be of type decimal and is returned here.

Time to see a few more more methods and then write your own!

**How do I use multiple methods?**

[How do I use multiple methods?](https://dmacc.blackboard.com/webapps/blackboard/content/listContent.jsp?course_id=_102593_1&content_id=_7286947_1)

Recall the program with all code in the Main method:

class Program

{

public static void Main()

{

const decimal TAX\_RATE = 0.07m;

// Calculate book tax

decimal bookPrice = 25.99m; decimal bookTax;

bookTax = bookPrice \* TAX\_RATE:

Console.WriteLine("Book tax is {0:C}", bookTax);

// Calculate meal tax

decimal mealPrice = 12.39m; decimal mealTax; mealTax = mealPrice \* TAX\_RATE;

Console.WriteLine("Meal tax is {0:C}", mealTax);

}

}

It's true that is this a short program and very easy to read. The idea that code should also be scalable and reuseable comes into play even when starting small. Adding a method to CalculateTax() allows the programming to add as many needed tax calculations in Main and simply calling CalculateTax(). Did you also notice the use of a constant? It's important to continue using the skills you have learned moving forward.

Now, you can add a method to print the meal tax as currency. Notice, there is no message to the user of DisplayCurrency that it is printing a meal tax. That way, you can also use the method to print mealTax, bookPrice, candyPrice, or any price. This method can be used again and again, anytime the need to display currency arises. You won't need to think "How does that format string work again" as you add to your program, an added bonus! If you are not general in this method, you would need to write multiple methods, one for each type of displayed currencies (mealTax, bookPrice, candyPrice, ...).Multiple specific methods would defeat the intend of having fewing lines of line of code and having the ability to reuse the method.

private static void DisplayCurrency(decimal dollarAmount)

{

Console.WriteLine("{0:C}", dollarAmount);

}

Notice to get the output on one line, in the Main, there is Console.Write() instead of Console.WriteLine(). Formatting is very important to have meaningful output. Also, telling the user what the numbers represent is important as well. You practiced this in asking for user input; you prompted the user for the type of information (age, price, ...) and format (must be an integer, must be positive, ...).

class Program

{

private static decimal CalculateTax(decimal price, decimal taxRate)

{

decimal tax;

tax = price \* taxRate;

return tax; }

private static void DisplayCurrency(decimal dollarAmount) {

Console.WriteLine("{0:C}", dollarAmount); }

public static void Main() {

const decimal TAX\_RATE = 0.07m;

// Calculate book tax

decimal bookPrice = 25.99m; decimal bookTax;

bookTax = CalculateTax(bookPrice, TAX\_RATE);

Console.Write("Book tax is ");

DisplayCurrency (bookTax);

// Calculate meal tax

decimal mealPrice = 12.39m; decimal mealTax;

mealTax = CalculateTax(mealPrice, TAX\_RATE);

Console.Write("Meal tax is ");

DisplayCurrency (mealTax);

}

}

What is the return of DisplayCurrency? Why?

Time to apply your method basics knowledge of value returning methods (they return a value) and non-value returning methods (they do not return a value and are of return type void).

**Program with a method**

[Program with a method](https://dmacc.blackboard.com/webapps/blackboard/content/listContent.jsp?course_id=_102593_1&content_id=_7286947_1)

Here is a entire program that shows the placement of the method in the Program. It is contained in the class. It need declared before the Main, but following convention, you should declare methods above your Main.

using System;

using System.Data;

namespace Methods

{

class Program

{

private static void DisplayMessage()

{

Console.WriteLine("Print a message");

}

static void Main(string[] args)

{

DisplayMessage(); //Method call

}

}

}

**Method Format**

[Method Format](https://dmacc.blackboard.com/webapps/blackboard/content/listContent.jsp?course_id=_102593_1&content_id=_7286947_1)

Example of a method with no return type (uses void) and no arguments:

private static void DisplayMessage()

{

Console.WriteLine("Print a message");

}

A method contains an access modifier, a return type, a name, (argument list)

Text

Description automatically generatedAccess modifiers determine the visibility of the method public, private, protected

The keyword static is present for methods that are not class methods (more that later as well). It simply means the method behavior doesn't change (it's static). For now, the method you will write are all static.

Return types are data types or void if no value is being return.

Method names should be descriptive and follow the same rules as variables. However, Pascal Case is the convention for methods, to make them distinguishable from variable names. Pascal Case is similar to camelCase; however, the first letter is always a capital: PascalCase, camelCase.

An argument list is the list of values and their local variable names that exist only during the method's execution. If there are no arguments, then the argument list is empty parentheses ()

Example of a method with one argument:

private void DisplayMessage(String msg)

{

Console.WriteLine(msg);

}

More on the argument list in the next topic!

The return type you of the method above was void. Return types can be built-in types and objects as well:

* + int
  + float
  + double
  + decimal
  + bool
  + *... and others*
  + String  // class object
  + Dog    // class object
  + Car     // class object
  + *... and others*

More on return types in the next topic, too, Keep going!