



Module 1 Day 3

Expressions, Statements, Blocks and Branching

What makes an application?

- Program Data

- ✓ Variables & .NET Data Types

- ☐ Arrays

- ☐ More Collections (list, dictionary, stack, queue)

- ☐ Classes and objects (OOP)

- Program Logic

- Statements and expressions

- Conditional logic (if)

- ☐ Repeating logic (for, foreach, do, while)

- Methods (functions / procedures)

- ☐ Classes and objects (OOP principles)

- ☐ Frameworks (MVC)

- Input / Output

- User

- ☐ Console read / write

- ☐ HTML / CSS

- ☐ Front-end frameworks (HTML / CSS / JavaScript)

- Storage

- ☐ File I/O

- ☐ Relational database

- ☐ APIs

Git config correction

- `git config branch.main.mergeOptions "--no-edit"`

Statements

- The **actions that a program takes** are expressed in statements. Common actions include declaring variables, assigning values, calling methods, looping through collections, and branching to one or another block of code, depending on a given condition
- <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/statements-expressions-operators/statements>

Statement Blocks (code blocks)

- Multiple statements grouped together as a block
- { } delimit a “code block”
- Blocks can be nested within blocks through many levels
- Variable scope
 - Variable is “in scope” until the block it was declared in exits
 - Inner blocks can access variables declared in outer blocks
 - Not vice-versa

Methods

- A method is a *statement block* with a name
- Can be called from other code
- We can pass values into the method
- The method may return a value to the caller
- So far we have only *written* one method
 - Main method in Program.cs
- But we *have* called another method
 - Do you know what method we have been calling?
- AKA functions, subroutines in other languages

Methods

- Method header
 - Access modifier
 - Method return type
 - Any data type or “void”
 - Method parameters (zero or more of these):
 - Data type
 - Parameter name
- Method Body
 - The “statement block”
 - Return statement(s)

	Return Type	Method Name	Parameter List	
<code>public</code>	<code>int</code>	<code>MultiplyBy</code>	<code>(int multiplicand, int multiplier)</code>	<code>{</code>
			<code>int result = multiplicand * multiplier;</code>	
			<code>return result;</code>	
				<code>}</code>

Let's
Code

Calling Methods

- Call (aka Invoke) a method

```
int product = MultiplyBy(100, 30);
```

- Pass in parameters (arguments)

- Can be literal (as above), variable names, or expressions
- Variable names do not need to match (they are matched by position)
- But they *do* have to be compatible types

```
int width = 12;  
int length = 20;  
int area = MultiplyBy(width, length);
```

- Method calls may be embedded inside expressions!



Let's
Code

Boolean Expressions

- An expression which resolves (evaluates) to a Boolean value (T/F)

- Comparison

- ==, !=, <, <=, >, >=

- Examples

- (age >= 18)
 - (day == 1)
 - (speed > speedLimit)

Operator	Meaning
==	Equal To
!=	Not Equal To
>	Greater Than
<	Less Than
>=	Greater Than or Equal To
<=	Less Than or Equal To

Boolean Expressions – Logical Operators

- Expressions can be combined using Logical Operators

Logical Operator	Name	Example
&&	AND	A && B T if both A and B are T
	OR	A B T if either A or B or both are T
!	NOT	!A If A is T, !A is F, and vice-versa
^	XOR Exclusive OR	A ^ B T if either A or B is T, but NOT if both are T (A && !B) (!A && B) (A B) && (!A !B) (A != B)

- Precedence

- !, ^, &&, ||
- Just use parentheses!

Conditional Code

- if (Boolean expression)
 statement-block
- if (Boolean Expression)
 statement-block
else
 statement-block
- if (Boolean Expression)
 statement-block
else if (Boolean Expression2)
 statement-block
- if (Boolean Expression)
 statement-block1
else if (Boolean Expression3)
 statement-block2
else
 statement-block3



Let's
Code

Bonus: Ternary Operator

```
int number = 3;
string backgroundColor;
if (number % 2 == 0)
{
    backgroundColor = "gray";
}
else
{
    backgroundColor = "white";
}
```

```
int number = 3;
string backgroundColor = number % 2 == 0 ? "gray" : "white";
```