OBJECTIVES

- What is a statement?
- What is an expression?
- What is a block of code?
- Comparison & logical operators

URLs:

- If-else.jpg
- if-elseif-ladder.jpg

A note about being human. (food, water, rest)

Open VisualStudio

Debugging!

Talk about how code can be placed in many different files

Open Program.cs

Today we are going to talk about COMPARISON, LOGIC and METHODS

EXPRESSIONS

```
/**
 * EXPRESSION: Expressions are variables, operators & methods that evaluate to a single value
 *  -each expression is evaluated separately, even if they are all in the same line
 *
 */
```

STATEMENTS

a line of code for the computer to execute
a command from a recipe, like "boil a pot of water"

```
 * STATEMENT: A complete unit of execution
 *  -a "sentence" or command
 *  -must end in a semi-colon;
```

Some statements

```
int x = 5 + 1;  //5+1 is an expression. The whole line is a statement
int z = (5 + 10) / 3;   //multiple expressions. 5+10 is calculated first
```

We use Logic operators with expressions

Comparison Operators

```
/**
 * COMPARISON OPERATORS: Compares values and returns TRUE or FALSE
 *  OPERATOR    MEANING
 *  ==          equal to
 *  !=          not equal
 *  >           greater than
 *  <           less than
 *  >=          greater than or equal to
 *  <=          less than or equal to
 */

Console.WriteLine(1==2);
Console.WriteLine(1 != 2);
Console.WriteLine(1>2);
```

Add (and)

```
//( ) allows for grouping. Things in parenthesis are executed and evaluated first
Console.WriteLine("1<2 : " + (1<2));
Console.WriteLine("1>=2" + (1>=2));
Console.WriteLine("1<=2" + (1 <= 2));
```

LOGICAL OPERATORS

```
/**
 * LOGICAL OPERATORS
 *  !   not
 *  &&  and
 *  ||  or
 *  ^   1 true, 1 false. AKA, XOR, exclusive OR
 */
```

Examples

```
Console.WriteLine("!true: " + (!true));
Console.WriteLine("true && true: " + (true && true));
Console.WriteLine("true && false: " + (true && false));
Console.WriteLine("true || false: " + (true || false));
Console.WriteLine("true ^ false: " + (true ^ false));
Console.WriteLine("true ^ true: " + (true ^ true));
```

TRUTH TABLE

```
/*
 * Truth table:

A      B         !A     A && B  A || B  A ^ B
TRUE   TRUE      FALSE  TRUE    TRUE    FALSE
TRUE   FALSE     FALSE  FALSE   TRUE    TRUE
FALSE  TRUE      TRUE   FALSE   TRUE    TRUE
FALSE  FALSE     TRUE   FALSE   FALSE   FALSE
*/
```

LOGIC

We can use IFs to control program flow

```
/**
 * IF statements use BOOLEAN expressions to control program flow
 *  -The expression goes in parenthesis
 *  -the conditional code to execute goes in {  }
 */
int cupsOfCoffee = 5;
if (cupsOfCoffee <2) {
    Console.WriteLine("You don't need a beverage carrier");
}
```

-- IF-ELSE image

ELSE

```
* ELSE executes if no other branch (IF/ELSEIF) executes
*  -this means "otherwise"

} else {
    Console.WriteLine("You should buy a large container of coffee");
}
```

-- IF-ELSEIF-ladder image

ELSE IF

*** ELSE IF allows for more branches**

```
        } else if (cupsOfCoffee < 8) {
            Console.WriteLine("You need a beverage carrier");
        }
```

Things to notice

*** Notice that only 1 branch of a IF-ELSEIF-ELSE tree can execute**

In ELSEIF

//Can be <8 because cupsOfCoffee <2 was already determined FALSE

BLOCKS

{ } allow you to group code

```
        //{  }  allow you to group lines of code
        {
            string a = "a";
            Console.WriteLine("a in a block {} of code: " + a);
        }
```

Scope: when a variable is no longer accessible… and not in memory

```
        //cannot access "a" because "a" disappeared when the code block ended
        Console.WriteLine("Can not get to A" + a);
```

METHODS

```
        /**
         * METHOD: a named block of code that you can execute/call multiple times
         *  -A method takes a specific input and produces a specific output
         *  -the code to execute goes inside the braces { }
         *  -Method signature:  [ACCESS_MODIFIER] [RETURN_TYPE] [METHOD_NAME] ([PARAMETER
LIST]) {
         *      public int addNumbers (int num1, int num2) {}
         *
         *  -return void if nothing is to be returned
         *  -parameters have a type and a name, separated by commas
         *  -Method names always start with a capital letter in C#
         */
```

Methods are actions that can be performed, often named with verbs

Remember functions in math class? f(x,y) => x*y

More to come later

Write a method

```
int multiplyTwoNumbers (int num1, int num2) {
    int product = num1 * num2;
    return product;
}
```

Call the method

```
int length = 4;
int width = 5;
int area = multiplyTwoNumbers(length,width);
Console.WriteLine("The area is " + area);
```

Need **STATIC** key word…. More on this in a week or two, just bare with me.

DEBUG!!!

The parameters are passed in order

`int area = multiplyTwoNumbers(width,length);`

DEBUG!!!

Go to lecture
Start at top

Show TEST EXPLORER
*what are tests? Code that runs other code to see if it works

How to run tests: run 1 vs run all

Walk through each class/method in detail…

1..2..3

4: After return type changes to DOUBLE, what happens if we return 2 vs 2.0?

6: IF statement, use debugger!

Use in Program.cs

```
LectureExample lectureExample = new LectureExample();
```

8: Input parameters… debug!

9: write in 1 line

```
return number > 5;
```

10: stacking conditional logic

```
if (addThree)
{
    number = number + 3;
}

// Why can't we use an else here for addFIVE?

if (addFive)
```

11: IF and Return

```
string returnValue = "";
if (number == 3)
{
    returnValue = "Fizz";
}
return returnValue;
```

12: Ternary IF -- consider skipping

```
return (number==3 ? "Fizz":"");
```

13: IF/ELSEIF

```
string returnValue = "";
if (number == 3) {
    returnValue = "Fizz";
} else if (number == 5) {
    returnValue = "Buzz";
}

return returnValue;
```

Notice the returnValue variable is outside the IF

14:

```
if (number >= 18)
```

15: shorten the method

```
if (!(number < 18))
{
    return "Adult";
}
else
```

```
    {
        return "Minor";
    }
```

16: complicated IF

```
    string returnValue = "";

    if (number % 2 == 0 && number > 100 && number % 5 == 0)
    {
        returnValue = "Big Even Number";
    } else if (number > 100) {
        returnValue = "Big Number";
    }

    return returnValue;
```

SLEEPIN

```
    public bool SleepIn(bool weekday, bool vacation)
    {
        if (vacation) {
            return true;
        }else if (!weekday) {
            return true;
        }
        return false;
    }
```

Another way

```
    bool SleepIn = !weekday || vacation;
    return SleepIn;
```

Suggestion that if an exercise is too hard, move on to another and come back to it