FOR LOOPS

- Loops are often used in a program to execute a series of statements over and over again. One common loop type is the "for" loop. This loop is typically used to execute a set of statements a fixed number of times.
- All **For** loops are based on an integer index, which is a numeric variable (e.g. **int**). The index starts at some specified number and is then increased or decreased each time through the loop until the last test logical condition becomes **true**.
- If you know how many times you need to iterate on some code, you want to use Visual C# counting. Counting is useful for adding items to a list or perhaps summing a known number of values to find an average.

- The **INITIALIZATION** step is executed once and is used to initialize a counter variable.
- The **EXPRESSION** step is executed before each repetition of the loop. If expression is true, the code is executed; if false, the loop is exited.
- The **UPDATE** step is executed after each loop iteration. It is used to update the counter variable.

```
for (Initialization; Expression; Update)
{
    //statements to be executed within the loop go here
}
```

- The **INITIALIZATION** statement is simply an assignment statement that sets your index variable to an initial value. You may declare your index variable at the same time you initialize it. An initialization statement starting the index at 0 would look like this:

```
for (int index = 0;
```

- The **EXPRESSION** that comes next is a logical expression that evaluates to true or false based on the current value of the loop index variable. The loop will continue to execute the body statements while the test expression is **true**. For example, we would continue the loop while the index was less than 10 like this:

```
for (int index = 0; index < 10;
```

- The test condition will be evaluated at the start of every loop. Once the statement is false (index is greater than 9 in this example), the program will skip to the next line of the code after our statement block.

- **UPDATE** will update the loop's index variable. The assignment will happen after the body of the loop has been executed each time through the loop – typically increment or decrement by 1, 2, or some other amount. But you must do something, otherwise your test condition never has a chance to go false and you will be stuck in an **infinite** loop!

```
for (int index = 0; index < 10; index++)
```

- It is common practice to shorten the names of loop variables to just a single character like "i".

```
for (int i = 0; i < 10; i++)
```

EXAMPLE #1

- This example will show pop-up message boxes with the numbers 0, 1, 2 and 3.

```
//declare a for loop with index "i" starting at one,
//incrementing once each time through the loop, and
//continuing so long as i < 4
 for (int i = 1; i < 4; i++)
 {
     MessageBox.Show(i.ToString());
 }
```

EXAMPLE #2

- This example will show pop-up message boxes with the numbers 4, 2 and 0.

```
//declare a for loop with index "i" starting at 4,
//decrementing twice each time through the loop, and
//continuing so long as i >= 0
 for (int i = 4; i >= 0; i-=2)
 {
     MessageBox.Show(i.ToString());
 }
```

WHILE LOOPS

- This type of loop is used to execute a set of statements **WHILE** a condition is **true**. You may not know in advance how many times the loop will execute, but you know the condition under which it should continue executing.
- A while loop is based on a logical expression that evaluates to either true or false. As long as this expression evaluates to true, the loop will continue to execute. The **while** loop syntax looks like this:

```
while (condition)
{
    //statements to be executed within the loop go here
}
```

EXAMPLE

- This example will calculate the factorial of 5 (1 x 2 x 3 x 4 x 5 = 120).

```
int counter = 5;
int result = 1;

while (counter > 1)
{
    result = result * counter;  //multiply result by current counter
    counter--;          //IMPORTANT - decrements the value of the counter
}
```

DO...WHILE LOOPS

- This loop is very similar to the **while** loop, except that the logical expression is not tested until the end of the loop body. This means that our loop body will ALWAYS execute at least one time.
- A **do-while** loop is based on a logical expression that evaluates to either **true** or **false**. As long as the expression is true, the loop will continue to execute. The **do-while** syntax looks like this:

```
do
{
    //statements to be executed within the loop go here
}
while (condition);    //notice the semi-colon this time
```

EXAMPLE

- Using the same example as before, we know that the calculations will perform at least one pass of our while loop, we can use a **do-while** loop to accomplish the same thing.
- This example will calculate the same as before (1 x 2 x 3 x 4 x 5 = 120).

```
int counter = 5;
int result = 1;
do
{
    result = result * counter;  //multiply result by current counter
    counter--;  //IMPORTANT - decrement the value of the counter
}
while (counter > 1); //while not done
```

BREAKING FREE!!!

- Sometimes you may want to terminate your for loop early, no matter what the index variable or test condition say! You can use the "**break**" statement to immediately exit any loop.
- The break statement can be used in all loop types.

```
string target = "Paper or plastic?";
int firstSpace = -1;

//walk over each character in the target string
for (int i = 0; i < target.Length; i++)
{
    //check to see if this character is equal to space
    if (target[i] == ' ')
    {
        //save the index of the first space
        firstSpace = i;
        //exit loop, we're done!
        break;
```

INFINITE LOOP!!!

- The most common programming mistake with loops is the **infinite loop**. This condition is caused by failing to change the logical expression within the loop body. If your while logical expression is true, and you do nothing in your loop body to ever make the expression false, your program will be stuck in that loop forever!
- You are responsible for changing the result of your logical expression in your while loop body.

```
while (1 < 2)
{
    //this is an infinite loop because 1 is always less than 2
}
```