Loops - Introduction to Repetition

One problem that arises in programming is what do we do when we want to repeat something lots of times?

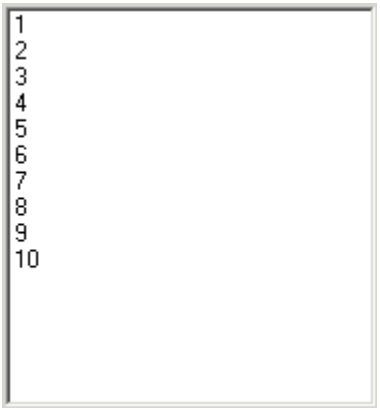
Consider the following example…

Let's say we wanted a list box to contain the numbers 1 to 10.

We could do the following…

```
lstTest.Items.Add(1);
lstTest.Items.Add(2);
lstTest.Items.Add(3);
lstTest.Items.Add(4);
lstTest.Items.Add(5);
lstTest.Items.Add(6);
lstTest.Items.Add(7);
lstTest.Items.Add(8);
lstTest.Items.Add(9);
lstTest.Items.Add(10);
```

Resulting in a list box looking like…



This is one way of tackling the problem, but what if we wanted the list to contain 100 entries? What about 1000 or more?

It isn't sensible to repeat lines of code like this, which is where loops come in.

Rather than us typing in the same commands over and over again, we program the computer to repeat commands until a suitable stopping point arises.

Types of Loops

Type 1 For-Next Loop

This loop repeats a section of code a specified number of times. So if we wanted to do a task 1000 times, this kind of loop is ideal.

Type 2 While Loop (Pre-processing)

This loop repeats a section of code until a certain condition is met, that is the condition may be met on the first attempt or on the thousandth, we simply don't know!

Type 3 Do Loop (Post-processing)

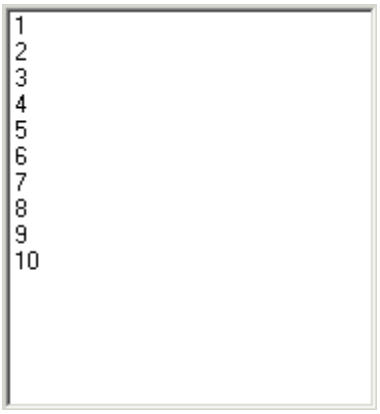
Always runs the code once then continues until a certain condition is met.

The While Loop

For any loop to work we need to know three things…

1. When the loop will start
2. When the loop will end
3. Where the loop is up to at a given point

Imagine we wanted to populate a list box with the numbers 1 to 10...



The while loop would look like this...

```
//var to store the counter
Int32 Index;
//initialise the counter at 1
Index = 1;
//while the counter is less than or equal to 10
while (Index <= 10)
{
    //add the counter to the list box (converting from Int32 to string)
    lstAddresses.Items.Add(Index.ToString());
    //increase the counter by 1
    Index++;
}
```

What we are doing here is firstly declaring a variable for the index of the loop.

We are stating that this index will start at a value of 1 and end at a value of 10 generating all values in between.

This code assumes a list box on the web page called lstAddresses.

The syntax of the loop is

```
while (Condition)
{
      //code to repeat goes here
}
```

Things to Watch with Do While Loops

What is wrong with the following loops?

```
//var to store the counter
Int32 Index;
//initialise the counter at 1
Index = 1;
//while the counter is less than or equal to 10
while (Index <= 10)
{
    //add the counter to the list box (converting from Int32 to string)
    lstAddresses.Items.Add(Index.ToString());
    //increase the counter by 1
    Index--;
}
```

```
//var to store the counter
Int32 Index;
//initialise the counter at 1
Index = 1;
//while the counter is less than or equal to 10
while (Index <= 10)
{
    //add the counter to the list box (converting from Int32 to string)
    lstAddresses.Items.Add(Index.ToString());
}
```

In both examples, the loop would never end. In the first example, Index is being decreased so will always be less than 10. In the second example Index is never changed and always remains at 1.

In the case of a do While loop you MUST make sure that the condition on which it runs becomes False so that the loop will not run forever.

Indexes Counters and Accumulators

Common mechanisms used with loops are indexes, counters and accumulators.

Indexes

In a loop an Index is a useful mechanism for identifying which item we are dealing with.

As we saw with data tables each record in the table is referenced by an index.

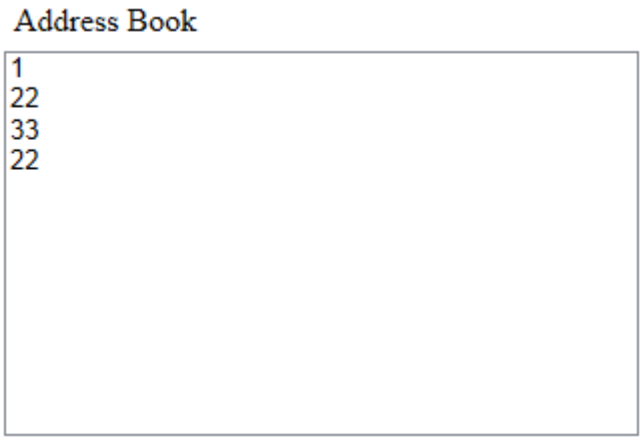
If our table contains the following data...

	AddressNo	HouseNo	Street	Town	PostCode	CountyCode	DateAdded	Active
▶	2	1	Some Street	Leicester	LE1 1BE	35	07/09/2012	True
	3	22	The Road	Nottingham	N19 6EF	48	07/08/2012	True
	4	33	High Street	Leicester	LE1 6FG	35	07/08/2012	True
	5	22	The Road	Nottingham	N19 6EF	48	07/08/2012	True
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

What will the following loop do?

```
//create an instance of the data connection class
clsDataConnection MyAddresses = new clsDataConnection();
//pass the parameter for post code
MyAddresses.AddParameter("@PostCode","");
//execute the query
MyAddresses.Execute("sproc_tblAddress_FilterByPostCode");
//var to store the index for the loop
Int32 Index = 0;
//var to store the count of records
Int32 RecordCount;
//var to store the HouseNumber
string HouseNumber;
//get the count of records
RecordCount = MyAddresses.Count;
//loop through all of the records
while (Index < MyAddresses.Count)
{
    //get the house number
    HouseNumber = MyAddresses.DataTable.Rows[Index]["HouseNo"].ToString();
    //add the new item to the list
    lstAddresses.Items.Add(HouseNumber);
    //increase the counter by 1
    Index++;
}
```

What we should see is a list box populated with four house numbers...



Counters

Counters may be used within loops to answer the question "How many?" Typically, we use them in While Loops when we have no idea how many times the loop will repeat.

Consider the following loop:

```
//var to store the LoanAmount
decimal LoanAmount;
//initialise loan amount at 20000
LoanAmount = 20000;
//while there is still a loan to pay off
while (LoanAmount >= 0)
{
    //make a payment
    LoanAmount = LoanAmount - 157;
}
```

Question, how many payments of £157 are required before the loan of £20,000 is paid off?

Yes, you could work it out on a calculator, but if we add a counter to the code we could work this out as follows.

```
//var to store the LoanAmount
decimal LoanAmount;
//var to store the counter
Int32 Counter = 0;
//initialise loan amount at 20000
LoanAmount = 20000;
//while there is still a loan to pay off
while (LoanAmount >= 0)
{
    //make a payment
    LoanAmount = LoanAmount - 157;
    //increase the counter by one
    Counter++;
}
```

Notice that we have added a new variable called Counter.

The variable has been initialised to zero via the line…

```
//var to store the counter
Int32 Counter = 0;
```

Also, each time the loop executes, the variable is incremented by one…

```
//increase the counter by one
Counter++;
```

Once the loop as completed, i.e. the loan is paid off, the Counter will contain a count of the number of instalments required to pay off the £20,000 loan.

Accumulators

Whilst counters answer the question of "how many items are there?" accumulators answer the question of "how much does a total come to?"

Look at the following code.

```
//var to store the interest
decimal Interest = 0;
//var to store Total Interest
decimal TotalInterest = 0;
//var to store the interest rate
decimal InterestRate = 0.004M;
//var to store the LoanAmount
decimal LoanAmount;
//var to store number of months (the counter)
Int32 LoanMonths = 0;
//initialise loan amount at 20000
LoanAmount = 20000;
//while there is still a loan to pay off
while (LoanAmount >= 0)
{
    //work out the interest to pay
    Interest = LoanAmount * InterestRate;
    //keep a track of how much interest has been paid so far (the accumulator)
    TotalInterest = TotalInterest + Interest;
    //make a payment adding on the interest for this month
    LoanAmount = LoanAmount - 157 + Interest;
    //increase the counter by one
    LoanMonths++;
}
```

In the above example, we are using the variable TotalInterest to "accumulate" the value of interest paid for the £20,000 loan.

By means of the counter LoanMonths we will know how many months the loan will take to pay off.