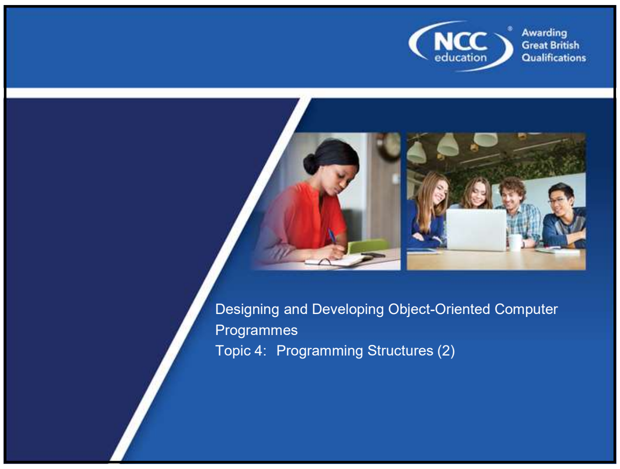
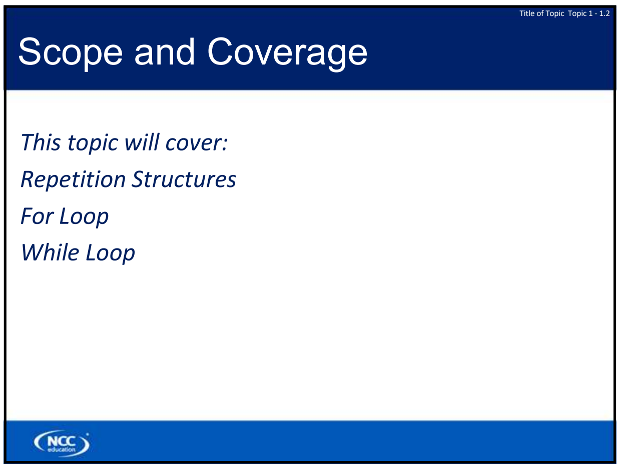


Awarding Great British Qualifications

Designing and Developing Object-Oriented Computer Programmes
Topic 4: Programming Structures (2)

1

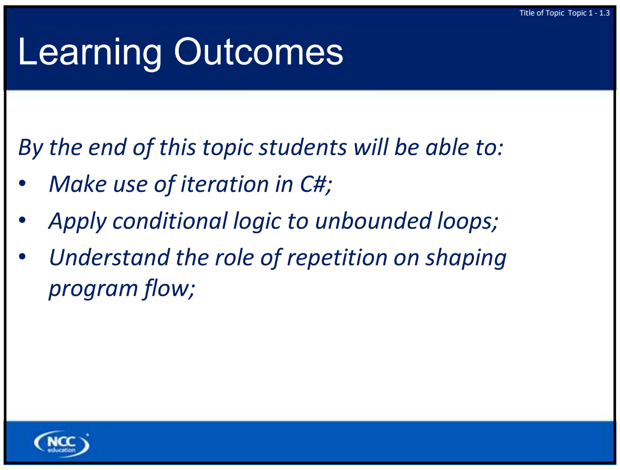


Title of Topic Topic 1 - 1.2

Scope and Coverage

This topic will cover:
Repetition Structures
For Loop
While Loop

2



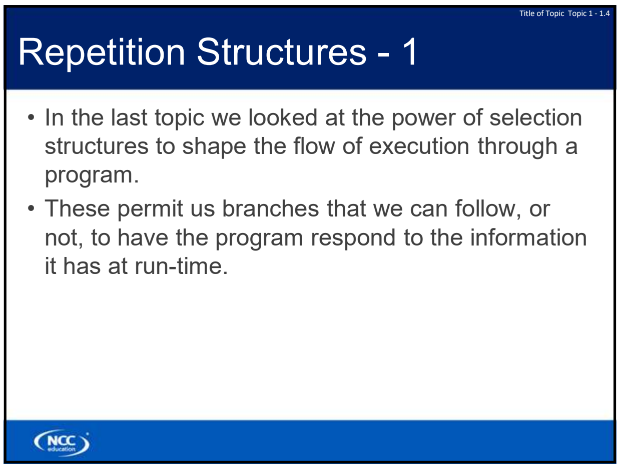
Title of Topic Topic 1 - 1.3

Learning Outcomes

By the end of this topic students will be able to:

- *Make use of iteration in C#;*
- *Apply conditional logic to unbounded loops;*
- *Understand the role of repetition on shaping program flow;*

3

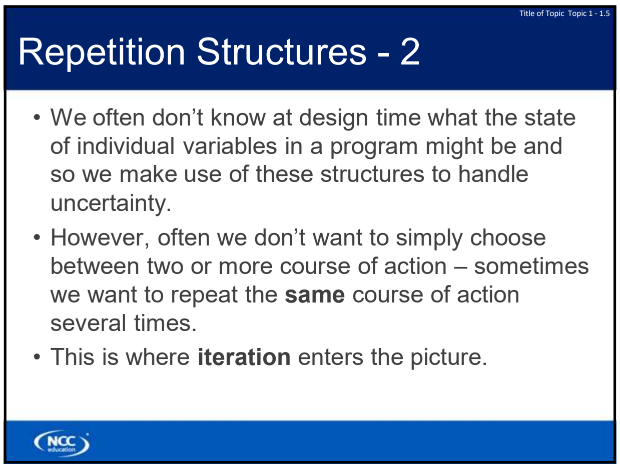


Title of Topic Topic 1 - 1.4

Repetition Structures - 1

- In the last topic we looked at the power of selection structures to shape the flow of execution through a program.
- These permit us branches that we can follow, or not, to have the program respond to the information it has at run-time.

4

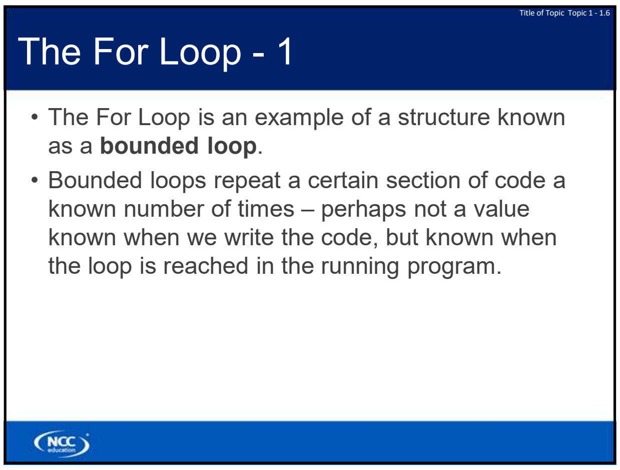


Title of Topic Topic 1 - 1.5

Repetition Structures - 2

- We often don't know at design time what the state of individual variables in a program might be and so we make use of these structures to handle uncertainty.
- However, often we don't want to simply choose between two or more course of action – sometimes we want to repeat the **same** course of action several times.
- This is where **iteration** enters the picture.

5

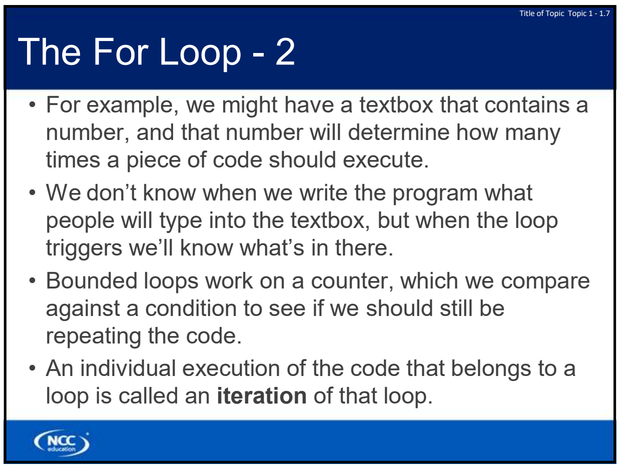


Title of Topic Topic 1 - 1.6

The For Loop - 1

- The For Loop is an example of a structure known as a **bounded loop**.
- Bounded loops repeat a certain section of code a known number of times – perhaps not a value known when we write the code, but known when the loop is reached in the running program.

6

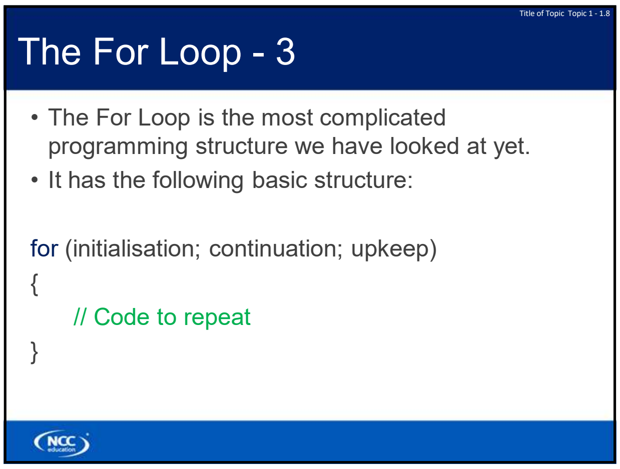


Title of Topic Topic 1 - 1.7

The For Loop - 2

- For example, we might have a textbox that contains a number, and that number will determine how many times a piece of code should execute.
- We don't know when we write the program what people will type into the textbox, but when the loop triggers we'll know what's in there.
- Bounded loops work on a counter, which we compare against a condition to see if we should still be repeating the code.
- An individual execution of the code that belongs to a loop is called an **iteration** of that loop.

7



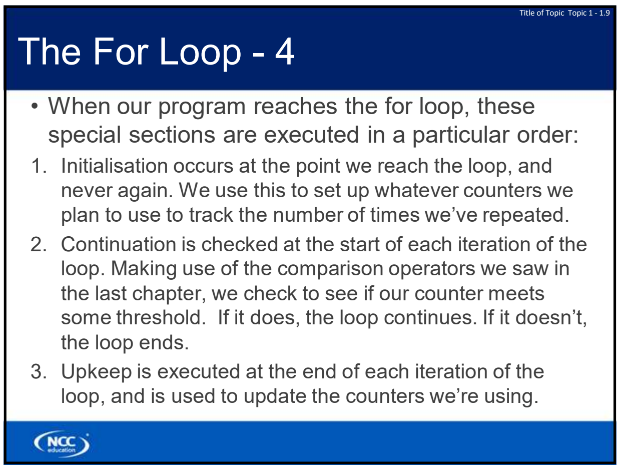
Title of Topic Topic 1 - 1.8

The For Loop - 3

- The For Loop is the most complicated programming structure we have looked at yet.
- It has the following basic structure:

```
for (initialisation; continuation; upkeep)
{
    // Code to repeat
}
```

8



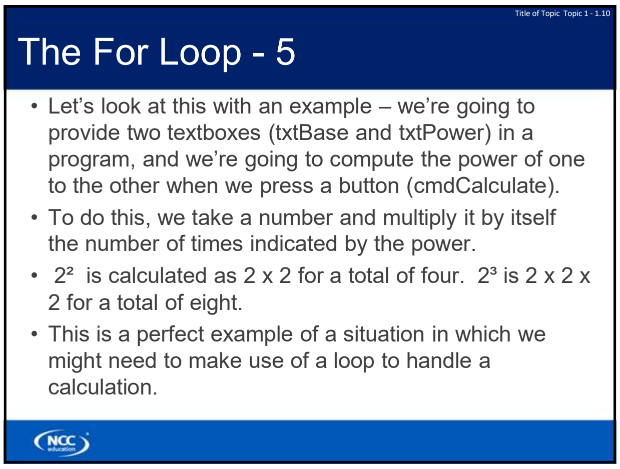
Title of Topic Topic 1 - 1.9

The For Loop - 4

- When our program reaches the for loop, these special sections are executed in a particular order:

1. Initialisation occurs at the point we reach the loop, and never again. We use this to set up whatever counters we plan to use to track the number of times we've repeated.
2. Continuation is checked at the start of each iteration of the loop. Making use of the comparison operators we saw in the last chapter, we check to see if our counter meets some threshold. If it does, the loop continues. If it doesn't, the loop ends.
3. Upkeep is executed at the end of each iteration of the loop, and is used to update the counters we're using.

9

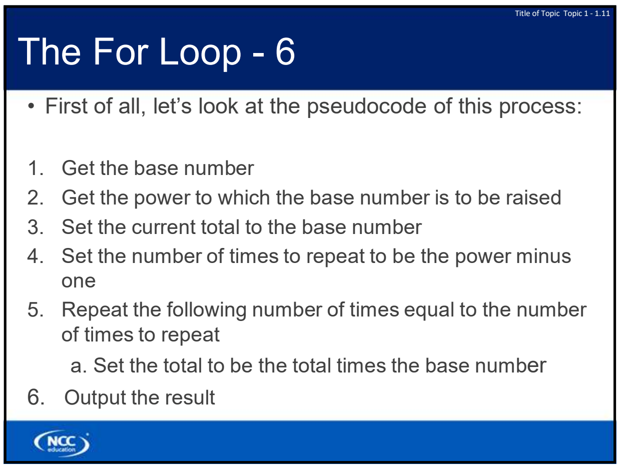


Title of Topic Topic 1 - 1.10

The For Loop - 5

- Let's look at this with an example – we're going to provide two textboxes (txtBase and txtPower) in a program, and we're going to compute the power of one to the other when we press a button (cmdCalculate).
- To do this, we take a number and multiply it by itself the number of times indicated by the power.
- 2^2 is calculated as 2 x 2 for a total of four. 2^3 is 2 x 2 x 2 for a total of eight.
- This is a perfect example of a situation in which we might need to make use of a loop to handle a calculation.

10



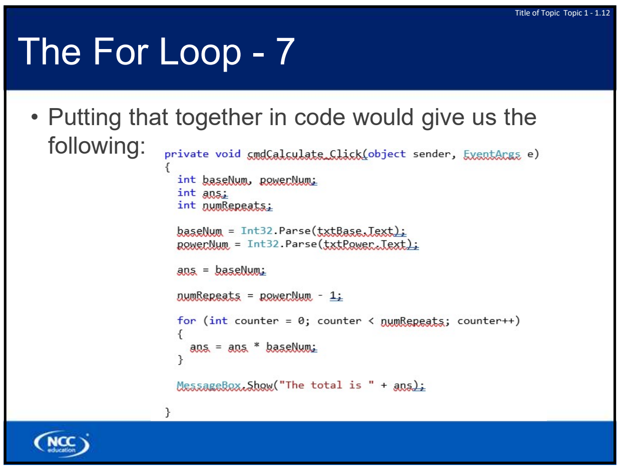
Title of Topic Topic 1 - 1.11

The For Loop - 6

- First of all, let's look at the pseudocode of this process:

```
1. Get the base number
2. Get the power to which the base number is to be raised
3. Set the current total to the base number
4. Set the number of times to repeat to be the power minus
   one
5. Repeat the following number of times equal to the number
   of times to repeat
     a. Set the total to be the total times the base number
6. Output the result
```

11



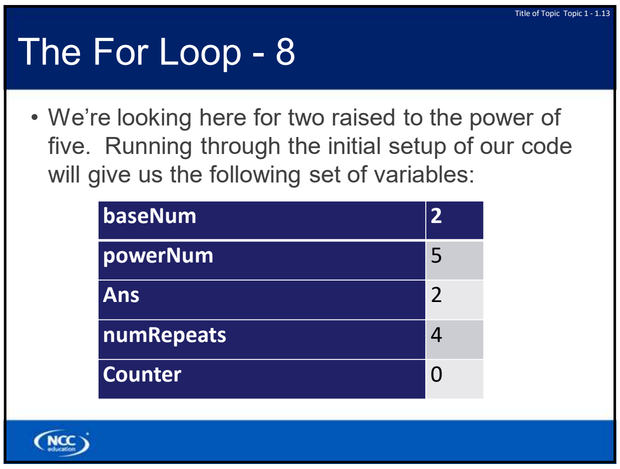
Title of Topic Topic 1 - 1.12

The For Loop - 7

- Putting that together in code would give us the following:

```
private void cmdCalculate_Click(object sender, EventArgs e)
{
  int baseNum, powerNum;
  int ans;
  int numRepeats;

  baseNum = Int32.Parse(txtBase.Text);
  powerNum = Int32.Parse(txtPower.Text);

  ans = baseNum;

  numRepeats = powerNum - 1;

  for (int counter = 0; counter < numRepeats; counter++)
  {
    ans = ans * baseNum;
  }

  MessageBox.Show("The total is " + ans);

}
```

12

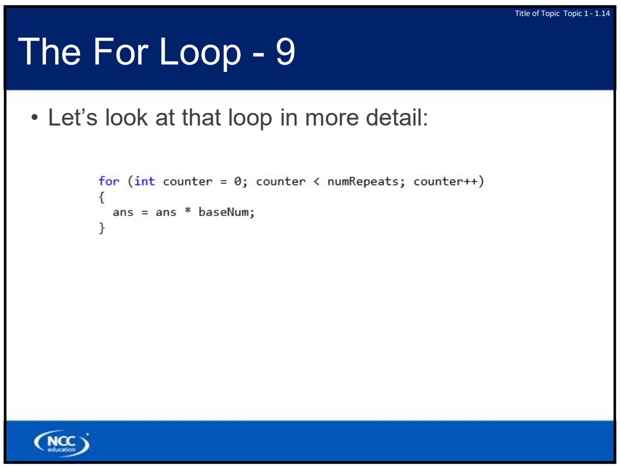


The For Loop - 8

- We're looking here for two raised to the power of five. Running through the initial setup of our code will give us the following set of variables:

baseNum	2
powerNum	5
Ans	2
numRepeats	4
Counter	0

13

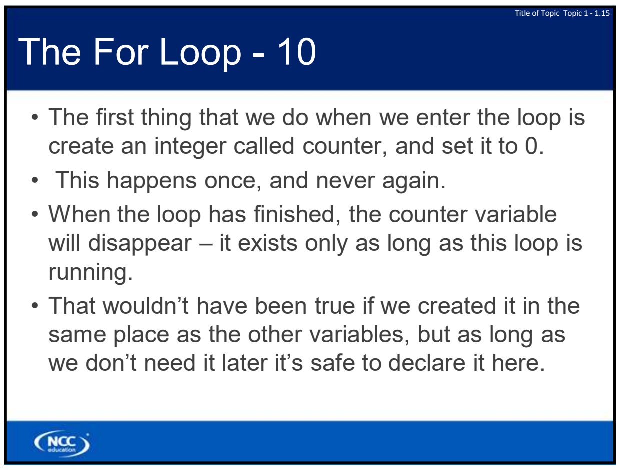


The For Loop - 9

- Let's look at that loop in more detail:

```
for (int counter = 0; counter < numRepeats; counter++)
{
  ans = ans * baseNum;
}
```

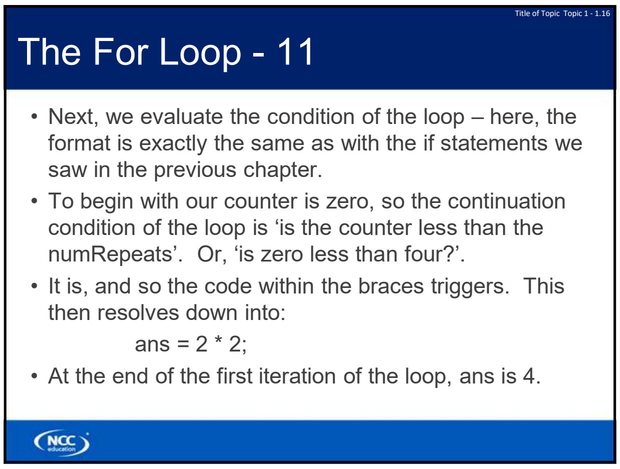
14



The For Loop - 10

- The first thing that we do when we enter the loop is create an integer called counter, and set it to 0.
- This happens once, and never again.
- When the loop has finished, the counter variable will disappear – it exists only as long as this loop is running.
- That wouldn't have been true if we created it in the same place as the other variables, but as long as we don't need it later it's safe to declare it here.

15



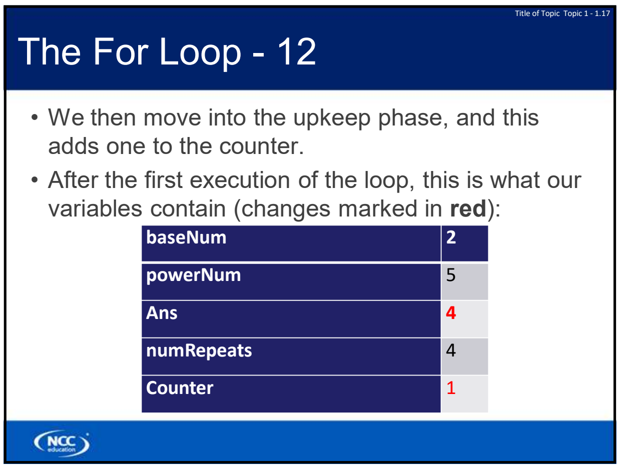
Title of Topic Topic 1 - 1.16

The For Loop - 11

- Next, we evaluate the condition of the loop – here, the format is exactly the same as with the if statements we saw in the previous chapter.
- To begin with our counter is zero, so the continuation condition of the loop is ‘is the counter less than the numRepeats’. Or, ‘is zero less than four?’.
- It is, and so the code within the braces triggers. This then resolves down into:

  ans = 2 * 2;
- At the end of the first iteration of the loop, ans is 4.

16



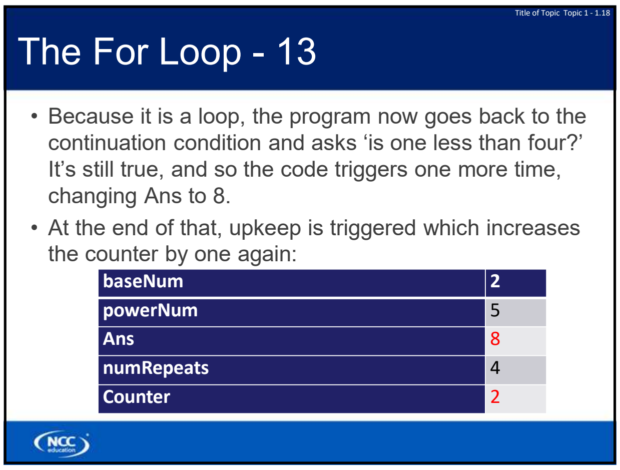
Title of Topic Topic 1 - 1.17

The For Loop - 12

- We then move into the upkeep phase, and this adds one to the counter.
- After the first execution of the loop, this is what our variables contain (changes marked in **red**):

baseNum	**2**
powerNum	5
Ans	**4**
numRepeats	4
Counter	1

17



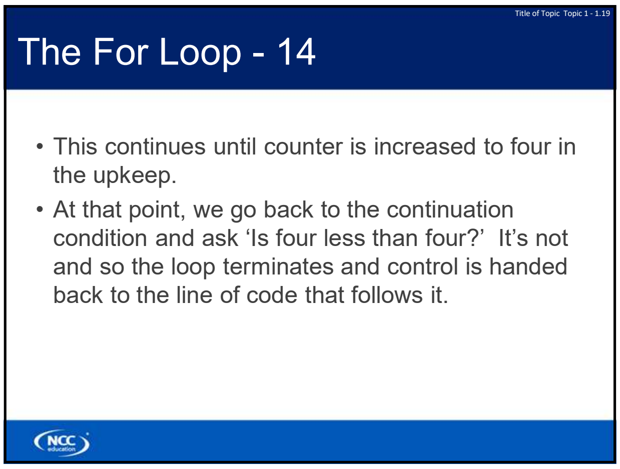
Title of Topic Topic 1 - 1.18

The For Loop - 13

- Because it is a loop, the program now goes back to the continuation condition and asks ‘is one less than four?’ It’s still true, and so the code triggers one more time, changing Ans to 8.
- At the end of that, upkeep is triggered which increases the counter by one again:

baseNum	**2**
powerNum	5
Ans	8
numRepeats	4
Counter	2

18

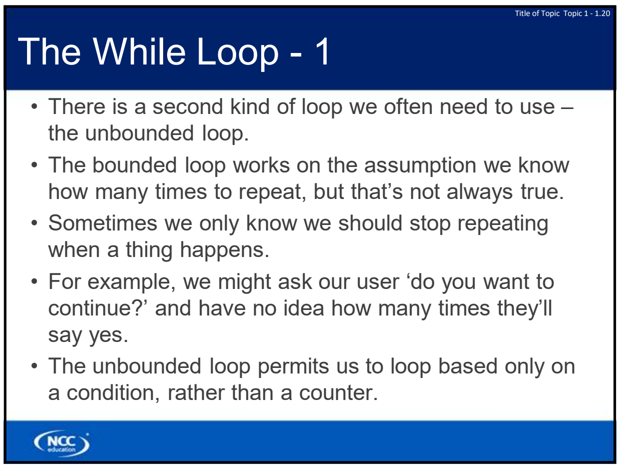


Title of Topic Topic 1 - 1.19

The For Loop - 14

- This continues until counter is increased to four in the upkeep.
- At that point, we go back to the continuation condition and ask ‘Is four less than four?’ It’s not and so the loop terminates and control is handed back to the line of code that follows it.

19

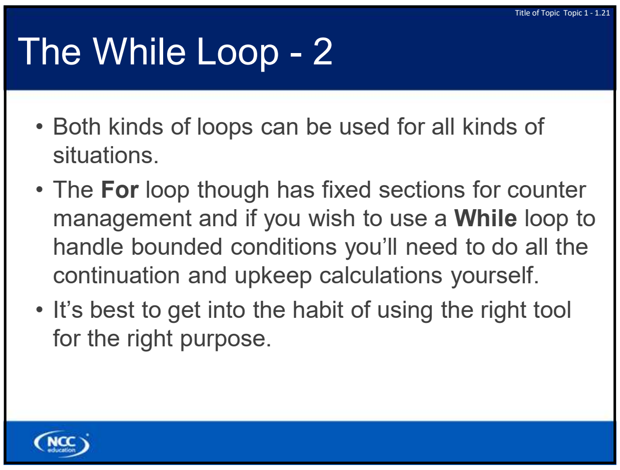


Title of Topic Topic 1 - 1.20

The While Loop - 1

- There is a second kind of loop we often need to use – the unbounded loop.
- The bounded loop works on the assumption we know how many times to repeat, but that’s not always true.
- Sometimes we only know we should stop repeating when a thing happens.
- For example, we might ask our user ‘do you want to continue?’ and have no idea how many times they’ll say yes.
- The unbounded loop permits us to loop based only on a condition, rather than a counter.

20

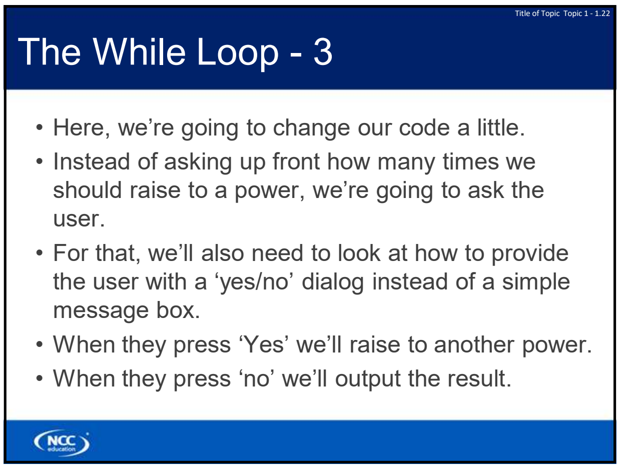


Title of Topic Topic 1 - 1.21

The While Loop - 2

- Both kinds of loops can be used for all kinds of situations.
- The **For** loop though has fixed sections for counter management and if you wish to use a **While** loop to handle bounded conditions you’ll need to do all the continuation and upkeep calculations yourself.
- It’s best to get into the habit of using the right tool for the right purpose.

21

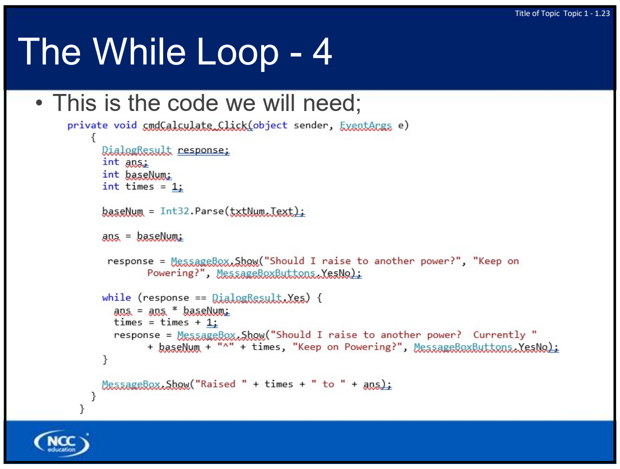


Title of Topic Topic 1 - 1.22

The While Loop - 3

- Here, we're going to change our code a little.
- Instead of asking up front how many times we should raise to a power, we're going to ask the user.
- For that, we'll also need to look at how to provide the user with a 'yes/no' dialog instead of a simple message box.
- When they press 'Yes' we'll raise to another power.
- When they press 'no' we'll output the result.

22



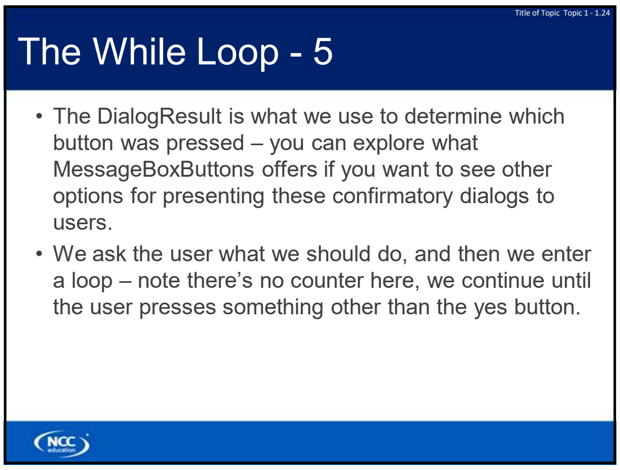
Title of Topic Topic 1 - 1.23

The While Loop - 4

- This is the code we will need;

```
private void cmdCalculate_Click(object sender, EventArgs e)
    {
      DialogResult response;
      int ans;
      int baseNum;
      int times = 1;

      baseNum = Int32.Parse(txtNum.Text);

      ans = baseNum;

       response = MessageBox.Show("Should I raise to another power?", "Keep on
             Powering?", MessageBoxButtons.YesNo);

      while (response == DialogResult.Yes) {
        ans = ans * baseNum;
        times = times + 1;
        response = MessageBox.Show("Should I raise to another power?  Currently "
              + baseNum + "^" + times, "Keep on Powering?", MessageBoxButtons.YesNo);
      }

      MessageBox.Show("Raised " + times + " to " + ans);
    }
  }
```

23



Title of Topic Topic 1 - 1.24

The While Loop - 5

- The DialogResult is what we use to determine which button was pressed – you can explore what MessageBoxButtons offers if you want to see other options for presenting these confirmatory dialogs to users.
- We ask the user what we should do, and then we enter a loop – note there's no counter here, we continue until the user presses something other than the yes button.

24

The While Loop - 6

- The while loop checks this continuation clause right at the very start, so we need to have it prepared for when it checks.
- Inside the loop, we ask the question again and again until the user decides to stop.
- We don't know when that's going to happen.

25

The While Loop - 7

- Note here the code is a little clumsy, repeating the same MessageBox code twice.
- There's a variation of this called the do while loop.

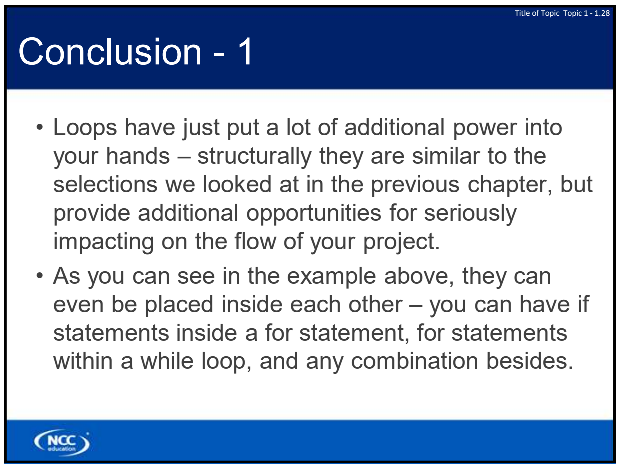
```
do
{
  response = MessageBox.Show("Should I raise to another power?  Currently "
        + baseNum + "^" + times, "Keep on Powering?", MessageBoxButtons.YesNo);

  if (response == DialogResult.Yes)
  {
    ans = ans * baseNum;
    times = times + 1;
  }
}
while (response == DialogResult.Yes);
```

26

The While Loop - 8

- For unbounded loops, if we want to iterate zero or more times, we use a while loop.
- If we want to iterate one or more, we use a do-while loop.
- Which is most appropriate will depend on your specific scenario.

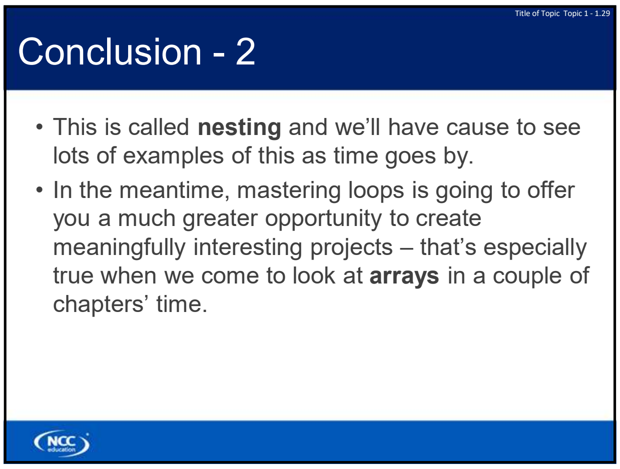
27



Conclusion - 1

- Loops have just put a lot of additional power into your hands – structurally they are similar to the selections we looked at in the previous chapter, but provide additional opportunities for seriously impacting on the flow of your project.
- As you can see in the example above, they can even be placed inside each other – you can have if statements inside a for statement, for statements within a while loop, and any combination besides.

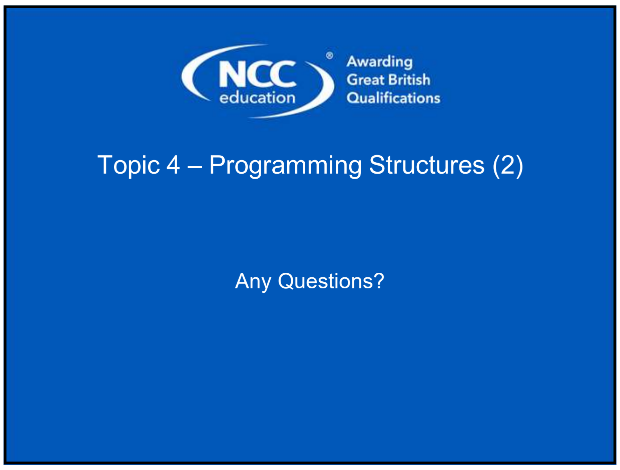
28



Conclusion - 2

- This is called **nesting** and we'll have cause to see lots of examples of this as time goes by.
- In the meantime, mastering loops is going to offer you a much greater opportunity to create meaningfully interesting projects – that's especially true when we come to look at **arrays** in a couple of chapters' time.

29



Awarding Great British Qualifications

Topic 4 – Programming Structures (2)

Any Questions?

30