THE FOR LOOP

The **for** statement is a looping structure that executes a set of statements a fixed number of times. In computer science, this is known as *iteration*. The **for** statement executes until a counter reaches an ending value. The **for** statement takes the following form:

```
for (counter = 1; counter <= 10; counter++)
{
        Statements you want to loop
}
```

There are 3 main parts of a **for** loop:

1. **Initialization** – the counter variable used to count the loop is set to an initial value. This can be set with a value or a variable.

2. **Boolean Expression** – controls the loop by testing the counter variable. If the expression is *true*, the loop runs again, if the expression is *false*, the loops stops running.

3. **Modifier** – this statement changes the counter variable after the statements in the loop finish. `counter++` has the same effect as `counter = counter + 1`.

Example 1: In this case, 10 alerts in all will be displayed showing the numbers 1 to 10.

```
for (counter = 1; counter <= 10; counter++)
{
      alert(counter);
}
```

Example 2: In this case, 5 alerts in all will be displayed showing even numbers counting down from 10 to 2.

```
for (i = 10; i >= 2; i = i - 2)
{
      alert(i);
}
```

Before moving on, experiment with different starting values, boolean expressions and modifier values. Try different loops that count up (increment) or down (decrement), as well as, a variety of operations (+,-,*,/).

INFINITE LOOPS

The condition of a loop is used to determine when the loop should stop executing. A for loop continues until its condition is false. What happens, though, if the condition never becomes false? The result is an infinite loop - one which continues forever.
A logic error can lead to an infinite loop. For example, the following statements create an infinite loop. Can you see why?

```
for (i = 10; i >= 2; i++)
{
      alert(i);
}
```

ACCUMULATOR PATTERN

Sometimes when we are iterating through a sequence of numbers, it can be useful for us to count something. The accumulator pattern allows us to keep a 'running total' as a loop runs. To do this, we must initialize a variable outside of the loop that will store the sum. As the loop runs, we will increment that sum as needed. After the loop finishes, it will store the result of our counting.
Here is an example:

```
var sum = 0;
var input = 0;
alert("Please enter 10 numbers, and I will tell you the sum!");

for (i = 1; i <= 10; i++)
{
      input = Number(prompt("Enter number " + i);
      sum = sum + input;
}

alert("Your sum is " + sum);
```

PRACTICE EXERCISES

1. **FOR COUNTER:** Use a for loop to output a range numbers determined from what the user enters.
    a. Every Number. If the user enters: 1 and 10, you would output 1,2,3,4 ..10.
    b. Even Numbers Only. If the user enters 1 and 10, you would output 2, 4, 6, …10.
    c. From highest to lowest. If the user enters 1 and 10, you would output 10, 9, 8, …1

2. **FOR SUM:** Write a for loop that will sum the numbers given a user entered start and stop value. Example: the user enters 5 and 9. You will add them =>5 + 6 + 7 + 8 + 9=35.

3. **FOR NUM WORDS:** Use a for loop to count the number of WORDS in a sentence.