Loops

Introduction

Loops allows use to repeat code without having to explicitly write the code. In terms of code path, loops makes our code go in circles. One example where we seen repeated code is in drawing an octagon.

Code Example 1: Ocatagon drawing code

```
ctx.beginPath();
ctx.lineTo(50,0);
ctx.rotate(45*Math.PI/180);
ctx.lineTo(50,0);
ctx.rotate(45*Math.PI/180);
ctx.lineTo(50,0);
ctx.rotate(45*Math.PI/180);
ctx.lineTo(50,0);
ctx.rotate(45*Math.PI/180);
ctx.lineTo(50,0);
ctx.rotate(45*Math.PI/180);
ctx.lineTo(50,0);
ctx.rotate(45*Math.PI/180);
ctx.lineTo(50,0);
ctx.rotate(45*Math.PI/180);
ctx.lineTo(50,0);
ctx.rotate(45*Math.PI/180);
ctx.lineTo(50,0);
ctx.rotate(45*Math.PI/180);
ctx.stroke();
```

The above example draws an octagon. Between `ctx.beginPath` and `ctx.stroke()` we repeated the same two lines of code 9 times. Loops provide a more convenient way to repeat such code. We will see a little later that using loops can make code more flexible as well.

For ... Loops

For loops are most often used to repeat code a known number of time. In the octagon we will know that we have to repeat our code 9 times.

The syntax for a for loop is as follows:

```
for([initiation]; [condition];[increment/decrement]){
  [code body/code to be repeated]
}
```

[initiation] : Code that runs once before the loop

[condition]: The condition is checked before the **code body** is executed. If the condition is **true** the code body is executed, otherwise the loop stops and the line after the **code body** is executed.
[increment/decrement]: Code that executes after every iteration of the loop.

The order in which the loop code runs is as follows

1. Run the initiation line
2. Check the condition
    a. If it's false stop the looping, and run the code after the loop
    b. If it's true goto step 3
3. Run the code body
4. No run the increment/decrement statement
5. goto step 2.

From the above outline, it is clear that if the condition is initially false, the loop will run a minimum of zero times.

We can then transform the Octagon drawing code to a shorter form, but using a loop.

Code Example 2: Using a loop to draw an octagon.

```
ctx.beginPath();
for(var i = 0; i < 9; i = i +1){
    ctx.lineTo(50,0);
    ctx.rotate(45*Math.PI/180);
}
ctx.stroke();
```

For the above code the `ctx.beginPath()` and `ctx.stroke()` statements are not inside the loop, because they only need to run once and not repeated.

The loop runs as follows:
1. `var i = 0;` executes creating the variable `i` and giving it a value of `0`.
2. next the line `i < 9` executes
    a. If it's false, the loop will ended.
    b. if it's true, we go onto to execute the body.
3. The two line in the body execute next, `ctx.lineTo(50,0)` and `ctx.rotate(45*Math.PI/180);`
4. The line `i = i +1` executes, next which changes the value of `i` by adding one to it. If the value of `i` never changed, the loop will never finish.
5. At this point we go back to step 2, and check the condition again. Keep in mind that **i** has now changed.

The above loop will then run until **i** reaches 9 and then run the ctx.stroke() command, which is after the loop.

Using a loop is an improvement over copying and pasting the code 9 times. Now, we only have two lines of drawing code to maintain, if we made a mistake we don't have to fix it 9 times, just once. The code is also shorter, so less frightening for students to read.

However we can do better. The above code only draw an octagon, or a fixed size. By changing the angle, size, and number of times we repeat we can write code that can draw regular polygons of any size.

CODE example 3:

```
var size = 50;
var sides = 8;
var angle = 360/sides;
ctx.beginPath();
for(var i = 0; i < (sides+1); i = i +1){
    ctx.lineTo(size,0);
    ctx.rotate(angle*Math.PI/180);
}
ctx.stroke();
```

In the above example we replaced 50 with the variable **size**, and the 45 with the variable **angle** inside the loop. We also replaced 9 in the condition with **sides +1**. Now if we can draw a polygon with any number of sides, or any size. We just need to change the appropriate variable. If we started want to draw an octagon, and decide want to draw a hexagon all wee need to do is change the number of sizes from 8 to 6. Write code in this way, is clearly more flexible and easier to maintain if we need to make revisions to our code.

The hexagon is an example of a problem whereby the code inside the loop did no depend on the loop index. No let's look at code that does depend on the loop index.

Let's try drawing 5 squares, starting at the origin (0,0), each box being 10 pixels in size, spaced 10 pixels between squares. That is the upper hand corners of the squares are 20 pixels apart.

CODE EXAMPLE 4: Using loop variables directly in the code body.

```
var c2CSpacing = 20;
var numSquares = 5;
var size = 10;
for(var  x=0; x < numSquares* c2CSpacing; x = x + c2CSpacing){
   ctx.fillRect(x,0,size,size);

}
```

In the above example we use the **x** variable as the x location of the square directly. Since the corners are spaced 20pixels are part, we add 20 each time with `x = x + c2CSpacing`.

We can also the loop variable indirectly. In the next example, we want to draw a set of squares diagonally.

CODE Example 5: Using the loop variable in the code body indirectly.

```
var ySpacing = 20;
var xSpacing = 20;
var size = 10;
var numSquares = 5;
for(var i =0; i  < numSquares; i = i+ 1){
   ctx.fillRect(i*xSpacing,i*ySpacing, size, size);
}
```

In the above example, **i** does not represent the x-location or the y-location. **i** represents the iteration number. The iteration number starts at 0, and goes to (numSquares -1). From the iteration number we can figure out the x and y location. AT iteration 0, we draw the square at 0,0. AT iteration 2 we draw at 20,20.

i	**i*xSpacing**	**i*ySpacing**
0	0	0
1	20	20
2	40	40
3	50	50
4	60	60

The above table represents all the values of **i** while running in the loop, as well as all the values of **i*xSpacing** and **i*ySpacing**. From the table we can expect the squares be drawn on a diagonal.

using loops to build strings

So far we have used loops for drawing. We can also loops to calculate values or create strings. For now we will focus on using loops to build a string.

A reminder that a string is a sequence of characters. We can make the sequence larger by concatenating more characters to it. The concept of string building with a loops is simply add to the string with each iteration of the loop. We can clearly combine this idea with conditional statement to build very sophisticated algorithms.

Let's build a simple list, using the <br> tag. HTML has a tags associated to lists, as we can also construct those list in the same way, but for now to keep things simple we will use the <br> tag. I want to insert the '2' multiplication table, as shown below.

2*1 = 1
2*2 = 2
etc..
I want each formula to appear on a separate line in HTML. To create this HTML snippet I can write the following loop.

```
var  string1 = "";
var  N = 10;
for(var i  =1; i <= N; i = i+1){
     string1 = string1 + 2 + "*" + i + "=" + 2*i + "<br>";
}
```

1. So we first start with an empty string.
2. On each iteration we add the formula to the string and a <br> tag to the string.
3. When the loop is done, we have a single that will have HTML code representing the '2' multiplication table.
4. When we assign this string to the **innerHTML** of an element, it will be displayed on the webpage.

Since after each iteration we add to the string, we call this process string building. We should also note that `string1` doesn't not have to be the first term after the assignment operator,'='.

For loop summary

For loops are used to repeat things a know amount of time. There is a large number of use of for loops. The above example we showed a small number of things that can be done, which included:

1. Using for..loops to repeat code a known number of times
2. Using for..loops to draw regular or repeated patterns
3. Using for..loops to build strings.

We will see more use of the for loops when we get to arrays!

The while loop

Like for loops while loops are used to repeat code. Although we can write for loops and while loops and while loops as for loops, the syntax of a while loop makes it more suitable for repeating code an unknown number of times.

One example use of this is pestering user for input until they give valid input. In this case, we do not know how many times a use will give invalid input.

The while loop looks like the following:

```
while([boolean condition]){
  [loop body/code to be executed]
}
```

[Boolean condition]: If the Boolean condition is true, the loop body will execute

The execution of the while loop is very straight forward

1) Check the Boolean execute the body, other stop and execute the line after the body.
2) Execute the body
3) Got to one.

It clear that the code inside the body, needs to eventually cause the boolean condition to be false, otherwise the loop will never end. A never ending loop is called an **infinite loop**.

In the next example, we use a while loop to prompt the user until they give us a number between 1 and 10;

Code Example 6:

```
var userString = "":
var numEntered= NaN;

while(isNan(numEntered)  ||
      numbersEntered < 1 ||
      numEntered > 10){
      userString = prompt("Please enter a number between 1 and 10");
    numEntered = Number(userString);
}

In the above example the statement
 isNan(numEntered)  || numbersEntered < 1 || numEntered > 10
```

checks for the opposite that we want. So if it's not a number, less than1, or greater than 10 the program should continue asking the user for input.