JavaScript

Repeating Things

Review of Monday’s Exercise: Step 1

Our task:

1. Ask user for column number
2. Ask user for color
3. Find table cell, change background color

```
var column = prompt("column number?");
var color = prompt("color?");
var selector = "td:nth-child("+ column + ")";
var cell = document.querySelector(selector);
cell.style.backgroundColor = color;
```

Review of Monday's Exercise: Step 2

Modification:

1. Ask user for column number
2. Check if number is a valid selection (we have 4 columns)
   If valid, proceed:
   a. Ask user for color
   b. Find table cell, change background color
3. Otherwise: display an alert

```
var column = prompt("column number?");
var color = prompt("color?");
var selector = "td:nth-child("+ column + ")";
var cell = document.querySelector(selector);
cell.style.backgroundColor = color;
```

```
var column = prompt("column number?");
if (column < 5 && column > 0) {
      var color = prompt("color?");
      var selector = "td:nth-child("+ column + ")";
      var cell = document.querySelector(selector);
      cell.style.backgroundColor = color;
}
else {
      alert("not a valid column number");
}
```

Review of Monday's Exercise: Step 3

New challenge: do this for 2 cells!

This is a very bad solution:

1. We violate the DRY principle!

2. What if we need to apply this to 10 elements? 100? 1000? etc...

```
var column1 = prompt("column number?");
if (column1 < 5 && column1 > 0) {
      var color1 = prompt("color?");
      var selector1 = "td:nth-child("+ column1 + ")";
      var cell1 = document.querySelector(selector1);
      cell1.style.backgroundColor = color1;
}
else {
      alert("not a valid column number");
}

var column2 = prompt("column number?");
if (column2 < 5 && column2 > 0) {
      var color2 = prompt("color?");
      var selector2 = "td:nth-child("+ column2 + ")";
      var cell2 = document.querySelector(selector2);
      cell2.style.backgroundColor = color2;
}
else {
      alert("not a valid column number");
}
```

Review of Monday’s Exercise: Step 4

1. Identify common functionality
2. "Factor out" this functionality into a new function
3. Call the function each time you need that functionality

We call this ***refactoring***:
changing (improving) your code
without changing what it does.

```
function getCell() {
    var column = prompt("column number?");
    if (column < 0 || column > 4) {
        alert("invalid selection");
    }
    else {
        var selector = "td:nth-child(" + column + ")";
        var cell = document.querySelector(selector);
        return cell;
    }
}

var cell1 = getCell();
cell1.style.backgroundColor = prompt("color?");

var cell2 = getCell();
cell2.style.backgroundColor = prompt("color?");
```

- What if we want to give our user a second chance? Or a third chance?
- How many times do we ask for a valid selection?

Introducing Repetition (Loops)

- Besides selecting which statements to execute, a fundamental need in a program is ***repetition***
- Repeat a set of statements under some conditions
- Between selection and repetition, we have the two most necessary programming statements

The Two Loops: ***while*** and ***for***

- The ***while*** loop repeats a set of statements while some condition is true.
    - Often called a ***sentinel*** controlled loop
    - while some *condition x* is true: execute block of statements inside { }
        - *condition x* is the sentinel

- The ***for*** statement is useful for iteration, moving through a sequence, one step at a time
    - Often called a count controlled loop
    - for a sequence of *n* steps: execute block of statements inside { }

The *while* loop

```
var x = 0;
while (x < 10) {
    console.log("x = " + x);  //write a message to the console (or do anything else!)
    x += 1;  //increment x; equivalent to this: x = x + 1
}
```

1. Test the condition. If the condition is true:
    a. Execute the statements inside the block { }
    b. Repeat (i.e., go back to 1.)
2. Otherwise: exit the loop

What will this code print out?

(ignore the last line in the console: it evaluates the last value of x)

The *while* loop

```
var x = 0;
while (x < 10) {
     console.log("x = " + x);  //write a message to the console (or do anything else!)
     x += 1; //increment x; equivalent to this: x = x + 1
}
```

Implementing a while loop:

1. Initialize the sentinel ***outside the loop***
2. Inside the loop, ***change something***: either the value of the sentinel variable or something else that will eventually lead to the condition being false and exiting the loop

The *for* loop

```
for (var i=0; i<10; i++) {
    console.log("i = " + i);  //write a message to the console (or do anything else!)
    //we DO NOT increment the counter: the for loop does it for us
}
```

We use 3 statements:

1. Initialize counter and give it a value **var i=0;**
   a. We use **i** for our counter by convention
2. State terminating condition **i<10;**
3. State how the counter changes after each iteration (or each step) **i++**
   a. i++ is equivalent to this: i = i + 1

When to use which?

- ***Use a for loop*** when we know in advance the number of iterations
  - Doing something n times (e.g., build a 8 x 8 table)
  - Iterate over a collection of HTML elements (e.g. , modify all links on a page)
- ***Use a while loop*** when we do not know in advance the number of iterations
  - Keep asking a user for valid input

  -
    ```
    var column = prompt("column number?");//initialize sentinel
    while (column < 0 || column > 4) {//test sentinel
          alert("invalid selection");
          column = prompt("column number?");//change sentinel
    }
    ```

What can go wrong?

What's the problem in this code?

```
var x = 0;
while (x < 10) {
    console.log("x = " + x);
}
```

What about this code?

```
for (var i=0; i>=0; i++) {
    console.log("i = " + i);
}
```

These are infinite loops: they go on forever (and will crash your browser)

A Useful Infinite Loop

- Our first attempt:

```
var column = prompt("column number?");//initialize sentinel
while (column < 0 || column > 4) {//test sentinel
      alert("invalid selection");
      column = prompt("column number?");//change sentinel
}
```

- A very different approach:

```
while (true) {
      var column = prompt("column number?");//initialize sentinel
      if (column > 0 && column < 5) {
            break; //breaks out of the loop at this point
      }
      alert("invalid selection");
}
```

- Use the break statement to end the loop prematurely

Loops & and the DOM

New method: document.**querySelectorAll**(selector)

- returns ***all*** elements that match the passed selector string
- the result is an array: a data type that holds a collection of some values
- Example:
    - var hobbits = ["Merry", "Pippin", "Frodo"];
- Accessible by index (0-based):
    - hobbits[0] contains the value "Merry"
    - hobbits[2] contains the value "Frodo"
    - hobbits[3] will cause an error
- We can also modify them: hobbits[0] = "Samwise Gamgee",
    - now hobbits[0] contains the value "Samwise Gamgee"

Loops & and the DOM

Array is a general-purpose data structure (we don't need the DOM to use it!)

It is essential in working with the DOM:

```
var cells = document.querySelectorAll("td");
//we can change any of the elements in the array:
cells[0].innerText = "new text here!";
```

Loops & and the DOM

We can also change all of the elements in an array:

```
var cells = document.querySelectorAll("td");
//an array knows its length: cells.length;
//which gives us:
for (var i=0; i<cells.length; i++) {
	cells[i].innerText = "new text here!";
}
```

[to be continued on Friday]