More Javascript Selection & Looping

If Statement

- Sometimes a block of code should only be run under certain conditions.
- Flow control — via if and else blocks — lets you run code if certain conditions have been met.
- While curly braces aren't strictly required around single-line if statements, using them consistently, even when they aren't strictly required, makes for vastly more readable code.

```
// Flow control
var foo = true;
var bar = false;

if (bar) {
  // this code will never run
  console.log("hello!");
}

if (bar) {
  // this code won't run
} else {
  if (foo) {
    // this code will run
  } else {
    // this code would run if foo and bar were both false
  }
}
```

Truthy and Falsy Things

- In order to use flow control successfully, it's important to understand which kinds of values are "truthy" and which kinds of values are "falsy."

```
// Values that evaluate to true
"0";
"any string";
[];  // an empty array
{};  // an empty object
1;   // any non-zero number
```

- Sometimes, values that seem like they should evaluate one way actually evaluate another.

```
// Values that evaluate to false
"";  // an empty string
NaN; // JavaScript's "not-a-number" variable
null;
undefined;  // be careful -- undefined can be redefined!
```

Switch Statement

- ‘Jump’ to a matching case statement.
- ‘Break’ to stop control from running on…

```
// A switch statement
switch (foo) {
  case "bar":
    alert("the value was bar -- yay!");
    break;
  case "baz":
    alert("boo baz :(");
    break;
  default:
    alert("everything else is just ok");
}
```

Loops

- Loops let a block of code run a certain number of times
- Note that in loops, the variable i is not "scoped" to the loop block even though the keyword var is used before the variable name.

```
// A for loop
// logs "try 0", "try 1", ..., "try 4"
for ( let i = 0; i < 5; i++ ) {
  console.log( "try " + i );
}
```

for loop

```
for ( [initialisation]; [conditional]; [iteration] )
{
 [ loopBody ]
}
```

- A for loop is made up of four statements and has structure shown above:
    - *initialisation statement*: executed only once, before the loop starts. It gives you an opportunity to prepare or declare any variables.
    - *conditional statement*: executed before each iteration, and its return value decides whether the loop is to continue. If the conditional statement evaluates to a falsey value, then the loop stops.
    - *iteration statement:* executed at the end of each iteration and gives you an opportunity to change the state of important variables. Typically, this will involve incrementing or decrementing a counter and thus bringing the loop closer to its end.
    - *loopBody statement:* runs on every iteration. It can contain anything. Typically, there will be multiple statements that need to be executed, and should be wrapped in a block ({...}).

For example

```
for ( [initialisation]; [conditional]; [iteration] ) {
 [ loopBody ]
}
```

```
//A typical for loop
for (let i = 0, limit = 100; i < limit; i++) {
  // This block will be executed 100 times
  console.log( 'Currently at ' + i );
  // Note: the last log will be "Currently at 99"
}
```

The while loop

- A while loop is similar to an if statement, except that its body will keep executing until the condition evaluates to false.

```
while ( [conditional] ) {
  [loopBody]
}
```

while example

- Notice that the counter is incrementing within the loop's body.

```
// A typical while loop
let i = 0;

while (i < 100) {
  // This block will be executed 100 times
  console.log("Currently at " + i);
  // increment i
  i++;
}
```

More while examples

- It's possible to combine the conditional and incrementer.
- Notice that the counter starts at -1 and uses the prefix incrementer (++i).
- These style is not very readable and should be avoided if possible

```
// A while loop with a combined conditional
// and incrementer
var let = -1;

while (++i < 100) {
  // This block will be executed 100 times
  console.log("Currently at " + i);
}
```

do-while

- This is almost exactly the same as the while loop, except for the fact that the loop's body is executed at least once before the condition is tested.

```
do {
  [ loopBody ]
} while ( [conditional] )
```

do-while example

```
// A do-while loop
do {
  // Even though the condition evaluates to false
  // this loop's body will still execute once.
  alert("Hi there!");
} while (false);
```

Breaking ...

- Usually, a loop's termination will result from the conditional statement not evaluating to true, but it is possible to stop a loop in its tracks from within the loop's body with the break statement.

```
// Stopping a loop
for ( var i = 0; i < 10; i++) {
  if (something)
  {
    break;
  }
}
```

Continuing...

- Continue the loop without executing more of the loop's body - the continue statement.

```
// Skipping to the next iteration of a loop
for ( let i = 0; i < 10; i++) {
  if (something)
  {
    continue;
  }
  // The following statement will only be executed
  // if the conditional 'something' has not been met
  console.log("I have been reached");
}
```