6 How to Write Conditional Code

Life is full of decisions. Code is the same way.

- If income < 10000, no tax is owed.
- If A's score > B's score, print "A wins!"

In JavaScript, as in most languages, you write conditional code using an IF statement. There are several forms.

To write a piece of code that is executed only if some situation is true, use the form

```
if (test) {
  code
}
```

The parentheses around the test are required. The curly braces are optional in some cases, but you should always include them.

A ***test*** is any JavaScript expression that returns a value. The IF will execute the code in the body of the IF if the value returned is true.

In JavaScript, as in many languages, true is defined as "not false." False in JavaScript is defined as any of the following:

- the pre-defined constants **false**, **null**, and **undefined**
- the result of a failed comparison, e.g., **x == y** or **x === y** where **x** and **y** are not equal, or **x < y** where **x** is not smaller than **y**
- the number 0
- the empty string

Anything else is considered true, including

- any number other than 0
- any object or array, even if empty
- the pre-defined constant **true**

For example, to add a number to an array of numbers, only when it is not already in that array, write this

```
if (numbers.indexOf(n) === -1) {
  numbers.push(n);
}
```

Quite often, you want to do one thing if a situation holds, or another thing if it doesn't. For that, use the form

```
if (test) {
  then-code
}
else {
  else-code
}
```

For example, suppose we need a function to generate a greeting for a letter. The input is a **person** object with their first and last names, their title, such as "Mr." or "Mrs.", and whether they are a friend of ours. The greeting should be just the first name for friends, and the full name, with title, for everyone else.

```
function getGreeting(person) {
  if (person.friend) {
    return "Dear " + person.firstName + ","
  }
  else {
    return "Dear " + person.title + " " + person.firstName + " " + person.lastName + ",";
  }
}
```

Finally, sometimes we have several different situations we need to handle. For that, use the form

```
if (test1) {
  code1
}
else if (test2) {
  code2
}
else if (test3) {
  code3
}
  ...
```

```
else {
  code when no test is true
}
```

For example, when a two-player game ends, there are three possible situations: the first player wins, the second player wins, or it's a tie. To create a string to show the first player, we could write

```
function getGameResult(score1, score2) {
  if (score1 > score2) {
    return 'You win!';
  }
  else if (score1 < score2) {
    return 'You lose!';
  }
  else {
    return 'It was a tie!';
  }
}
```

The Conditional Operator

Simple conditionals that just need to select a value can often be written more compactly, using the conditional operator. This is actually a pair of operators -- the question mark **?** and the colon **:**. For example, to do the example above, telling a player how they did, you could write one **return** statement with a conditional expression:

```
function getGameResult(score1, score2) {
  return (score1 > score2) ? 'You win!'
    : (score1 < score2) ? 'You lost!'
    : 'It was a tie!';
}
```

The syntax is *test* **?** *then-value* **:** *else-value*. In the example above, the else-value is itself another conditional expression.

How to Compare Things

In programming, comparison means asking if two things are equal or, if not, which one is bigger than the other.

Bigger and smaller are simple. To see if X is bigger than Y, just evaluate **x > y**. This is a **Boolean expression**. Like any expression, it returns a value. That value is either **true** or **false**.

Equality is a little trickier. There are two equality operators in JavaScript

- **x === y** tests for **strict** equality, i.e., equal values of the same type.
- **x == y** tests for **loose** equality, i.e., equal values, but with possible type conversion.

==, i.e., loose equality, was use in most early JavaScript code, but these days, **===**, i.e., strict equality, is recommended.

Here are common cases where **===** returns false, but **==** returns true. As you can see, many of these true values are unintuitive:

- 10 == "10"
- 0 == ""
- 0 == **false**
- 1 == **true**
- "" == **false**

For a complete list of the rules for loose equality, see this table.

Note that because JavaScript does not distinguish integers from floating point, **10 === 10.0** is true. This differs from C and Java.

To test if two things are not equal, use **!==**.

Boolean expressions and values are just like any other. A common mistake of novices is thinking that boolean expressions only work in IF statements. This leads them to write silly code like this:

```
function isAdult(person) {
  if (person.age >= 21) {
    return true;
  }
  else {
    return false;
  }
}
```

There is no need for an IF here. This function can and should be defined in one line:

```
function isAdult(person) {
  return person.age >= 21;
}
```

Boolean Operators

Although IF statements are all you need to do computational logic, often you can simplify your code by using boolean operators to combine true / false values. There are three boolean operators. The operators themselves are special characters, but are referred to by English names:

- **conjunction** or **and**: *value1* **&&** *value2* is true if both of the values are true, and false otherwise
- **disjunction** or **or:** *value1* **||** *value2* is true if either of the values are true, and false otherwise
- **negation** or **not: !** *value* is true if the value is false, and false if it is true

The **and** and **or** operators are infix operations, like addition (**+**), so you can write "... && ... && ..." and so on. If you mix **and** and **or**, e.g., "... && ... || ...", parentheses to be clear about how you want things combined.

For example, suppose we needed to test if an object **x** is an array with at least one element.. This function would work:

```
function isIndexable(lst) {
  return Array.isArray(lst) && lst.length > 0
}
```

An important property of boolean expressions is that evaluation goes left to right, and stops as soon as the result is determined. This is called **short-circuiting**. In the code above, if **lst** is not an array, that's a false value, so the whole expression has to be false. The rest of the expression is not evaluated. If short-circuiting did not happen, **isIndexable(null)** would trigger an error, rather than returning **false.**