JavaScript and jQuery

/* Week 4: JS Basics, Part 2 – You're HOW OLD? */

// Frontend Web Development, Part II
// Bloomington Code School – Spring 2016

LAST WEEK

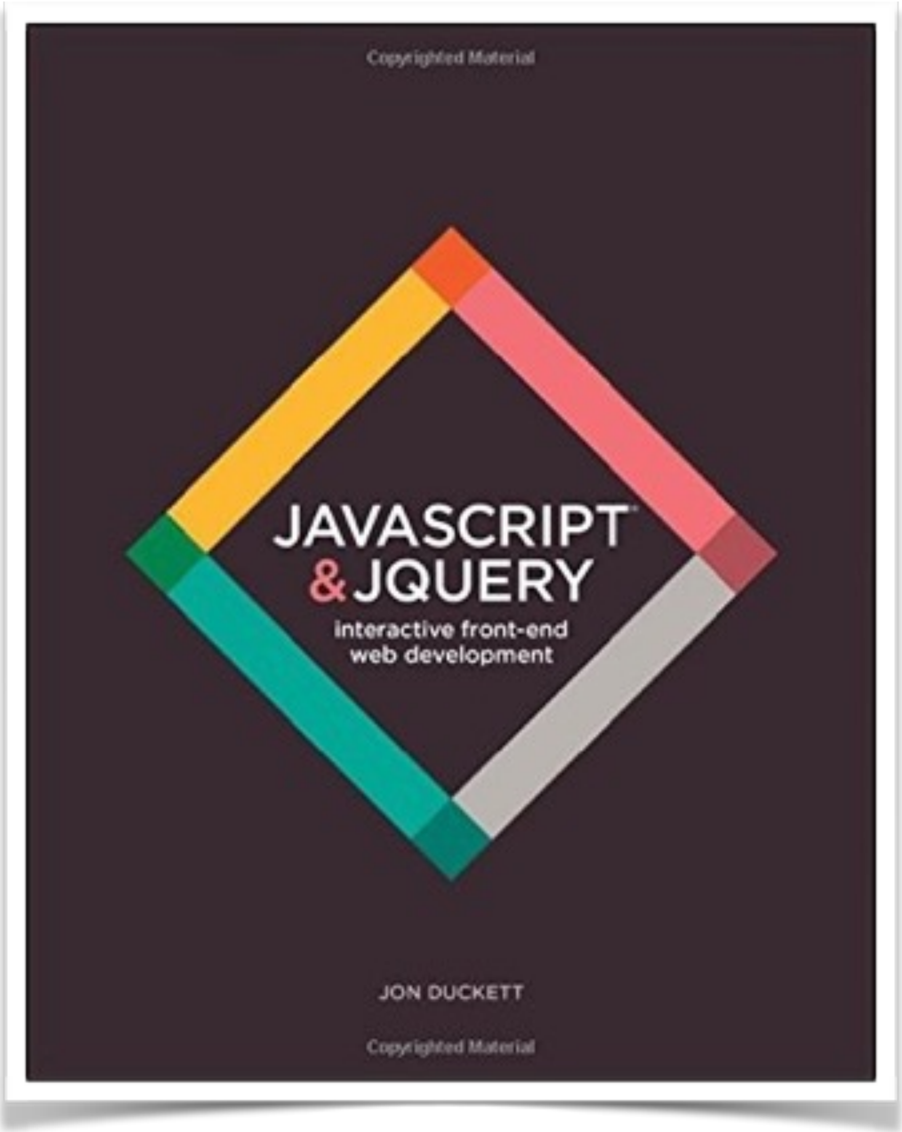
// Review Week 2 Challenge with code

// Chrome Developer Tools

THIS WEEK

// Operators & Conditions

// Discuss git & GitHub.com accounts



JavaScript and JQuery
by Jon Duckett

http://bit.ly/csjq1

A Smarter Way to Learn JavaScript
by Mark Myers

http://bit.ly/csjs1

A Smarter Way to Learn JavaScript
The new tech-assisted approach that requires half the effort
1 Read a 10-minute chapter of this book to get each concept.
2 Make the knowledge stick. Do the chapter's free interactive exercises at ASmarterWayToLearn.com.
Mark Myers

ORDER OF PRECEDENCE

```
// Parens
// In/De-crement
// Maths
// Assignments
```

CONDITIONS & COMPARISONS

COMPARISON OPERATORS

// COMPARE TWO VALUES [mostly numbers] [...mostly]

COMPARISON OPERATORS

```
// COMPARE TWO VALUES [mostly numbers] [...mostly]

// equal to: ( x === y )

// less than:  ( x < y )

// less than or equal to:  ( x <= y )

// greater than:  ( x > y )

// greater than or equal to:  ( x >= y )
```

IF STATEMENTS

```
// only IF the ( condition ) is true,
// THEN execute the { code block }


if ( x < 5 ) {
  // do stuff only if x is less than 5
}

if ( x >= 1 ) {
  // do stuff if x is greater than or equal to 1
}
```

IF...ELSE...

```
// only IF the ( condition ) is true,
// THEN execute the { code block }
// OTHERWISE execute the else { code block }

if ( x < 5 ) {
  // do stuff only if x is less than 5
} else {
  // do this only if x is NOT less than 5
}
```

IF...ELSE...IF...

```
// only IF the ( condition ) is true,
// THEN execute the { code block }
// OTHERWISE IF the else ( condition ) is true,
// THEN execute the else { code block }

if ( x < 5 ) {
  // do stuff only if x is less than 5
} else if ( x >= 0 ) {
  // do this only if x is NOT less than 5
  // AND only if x is greater than or equal to 0
}
```

IF...ELSE...IF...

```
// these could be broken up into two separate IFs
// instead of using the IF ELSE IF pattern
```

IF…ELSE…IF…

```
// these could be broken up into two separate IFs
// instead of using the IF ELSE IF pattern

if ( x < 5 ) {
  // do stuff only if x is less than 5
}

if ( x >= 5 && x >= 0 ) {
  // do this only if x is NOT less than 5
  // AND only if x is greater than or equal to 0
}
```

IF...ELSE...IF...

```
// these could be broken up into two separate IFs
// instead of using the IF ELSE IF pattern


// but WAIT!... what’s this?


if ( x >= 5 && x >= 0 ) {
  // do this only if x is NOT less than 5
  // AND only if x is greater than or equal to 0
}
```

LOGICAL OPERATORS

LOGICAL OPERATORS

```
// they reduce a statement to a BOOLEAN value
```

LOGICAL OPERATORS

```
// they reduce a statement to a BOOLEAN value
// [‘boolean value’ just means either true or false]
```

IF STATEMENTS *love*

LOGICAL OPERATORS

```
// they reduce a statement to a BOOLEAN value
// [‘boolean value’ just means either true or false]

// and remember:
// IF statements only execute IF ( true )
```

IF STATEMENTS *love*

LOGICAL OPERATORS

```
// they reduce a statement to a BOOLEAN value
// [‘boolean value’ just means either true or false]

// and remember:
// IF statements only execute IF ( true )
// so IF ( this condition is true )
```

IF STATEMENTS *love*

LOGICAL OPERATORS

```
// they reduce a statement to a BOOLEAN value
// [‘boolean value’ just means either true or false]

// and remember:
// IF statements only execute IF ( true )
// so IF ( this condition is true )
// THEN { this code block will be activated! }
```

IF STATEMENTS *love*

LOGICAL OPERATORS

```
// they reduce a statement to a BOOLEAN value
// [‘boolean value’ just means either true or false]

// and remember:
// IF statements only execute IF ( true )
// so IF ( this condition is true )
// THEN { this code block will be activated! }

// BUT IF ( the condition is false )
// then { this code is ignored :(  }
```

back to

LOGICAL OPERATORS

```
// they reduce a statement to a BOOLEAN value
```

LOGICAL OPERATORS

```
// they reduce a statement to a BOOLEAN value

// in natural language we do this with AND and OR
```

LOGICAL OPERATORS

```
// they reduce a statement to a BOOLEAN value

// in natural language we do this with AND

// EX: She likes Hot Cheetos AND Takis. ✅(it’s true!)
```

LOGICAL OPERATORS

```
// they reduce a statement to a BOOLEAN value

// in natural language we do this with AND

// EX: She likes Hot Cheetos AND Takis. ✅(it’s true!)
//
// it’s an accurate statement as long as she likes BOTH
// (She likes Hot Cheetos✅) AND (She likes Takis✅)
```

LOGICAL OPERATORS

```
// they reduce a statement to a BOOLEAN value

// in natural language we do this with AND

// EX: She likes Hot Cheetos AND Takis. ✅(it’s true!)
//
// it’s an accurate statement as long as she likes BOTH
// (She likes Hot Cheetos✅) AND (She likes Takis✅)
//
// if she didn’t like one, it’d be a false ❌ statement
```

LOGICAL OPERATORS

```
// they reduce a statement to a BOOLEAN value

// in natural language we do this with OR

// EX: She likes Hot Cheetos OR Takis. ✅(it’s true!)
//
// it’s an accurate statement if she likes EITHER ONE
```

LOGICAL OPERATORS

```
// they reduce a statement to a BOOLEAN value

// in natural language we do this with OR

// EX: She likes Hot Cheetos OR Takis. ✅(it’s true!)
//
// it’s an accurate statement if she likes EITHER ONE
// (She likes Hot Cheetos✅) OR (She likes Takis✅)
```

LOGICAL OPERATORS

```
// they reduce a statement to a BOOLEAN value

// in natural language we do this with OR

// EX: She likes Hot Cheetos OR Takis. ✅(it’s true!)
//
// it’s an accurate statement if she likes EITHER ONE
// (She likes Hot Cheetos✅) OR (She likes Takis✅)
//
// if she hates both, it’d be a false ❌ statement
// but if she likes one (or both), it’s true ✅
```

back to

LOGICAL OPERATORS

```
// they reduce a statement to a BOOLEAN value
```

LOGICAL OPERATORS

```
// they reduce a statement to a BOOLEAN value
// by using AND and OR
```

LOGICAL OPERATORS

```
// they reduce a statement to a BOOLEAN value
// by using  &&  for AND
//
//           (two ampersand symbols)
```

LOGICAL OPERATORS

```
// they reduce a statement to a BOOLEAN value
// by using  &&  for AND
//
//           (two ampersand symbols)

// by using  ||  for OR
//
//           (two pipe symbols)
```

LOGICAL OPERATORS

```
// they reduce a statement to a BOOLEAN value
// by first reducing each side of the AND or OR
```

LOGICAL OPERATORS

```
// they reduce a statement to a BOOLEAN value
// by first reducing each side of the AND or OR

// a quick real example:

var likesCheetos = true;
var likesTakis = false;

if (likesCheetos && likesTakes) {
  alert(‘She likes BOTH!’);
}
```

COMPARING
COMPARISON OPERATORS AND
LOGICAL OPERATORS

```
// similarly, they reduce a statement to a BOOLEAN value

// COMPARE TWO VALUES [mostly numbers] [...mostly]
// with any of the greater/less than comparisons:
//     <    >    <=    >=
```

COMPARING
COMPARISON OPERATORS AND
LOGICAL OPERATORS

```
// similarly, they reduce a statement to a BOOLEAN value

// COMPARE TWO VALUES [mostly numbers] [...mostly]
// with any of the greater/less than comparisons:
//     <    >    <=    >=

// COMPARE TWO VALUES [numbers and beyond!]
//
// use  ==  for EQUALITY
//
// use  ===  for STRICT EQUALITY
```

LOGICAL OPERATORS

```
// they reduce a statement to a BOOLEAN value
// by first reducing each side of the AND or OR or EQUALS

// a quick real example:

var likesCheetos = true;
var likesTakis = false;

if (likesCheetos === true) {
  likesTakis = true;
  alert(“If you like one, you must like both.”);
}
```

LOGICAL OPERATORS

```
// but what if we wanted to know:
//
//   * what is FALSE?
//       (a.k.a. NOT TRUE)
//
//   * or what is NOT EQUAL?
//       (a.k.a. INEQUALITY)
//
```

BANG!

```
// is easier than having to say: “exclamation point”
```

BANG!

```
// is easier than having to say: “exclamation point”

// anytime you see ONE BANG before a !value
// it means...
```

BANG!

```
// is easier than having to say: “exclamation point”

// anytime you see ONE BANG before a !value
// it means... the value after the bang will be
//             converted into a boolean, then the
//             opposite of that boolean is returned
```

BANG!

```
// is easier than having to say: “exclamation point”

// anytime you see ONE BANG before a !value
// it means... the value after the bang will be
//             converted into a boolean, then the
//             opposite of that boolean is returned

// so:  (!true)  is the same as  (false)
```

BANG!

```
// is easier than having to say: “exclamation point”

// anytime you see ONE BANG before a !value
// it means... the value after the bang will be
//             converted into a boolean, then the
//             opposite of that boolean is returned

// so:  (!true)  is the same as  (false)
// and: (!false) is the same as  (true)
```

BANG!

```
// is easier than having to say: “exclamation point”

// anytime you see ONE BANG before a !value
// it means... the value after the bang will be
//             converted into a boolean, then the
//             opposite of that boolean is returned

// so:  (!true)  is the same as  (false)
// and: (!false) is the same as  (true)
// AND: (!!true) is the same as  (true)
```

BANG!

```
// is easier than having to say: “exclamation point”

// anytime you see ONE BANG before a !value
// it means... the value after the bang will be
//             converted into a boolean, then the
//             opposite of that boolean is returned

// so:  (!true)  is the same as  (false)
// and: (!false) is the same as  (true)
// AND: (!!true) is the same as  (true)
// SO: !!!!true  is the same as  (true)
```

BANG!

```
// when you see multiple bangs!!!
// think of it as if parenthesis were involved

// SO: !!!!true    is the same as  (true)

//  !(!(!(!true))) is the same as  !!!!true  and  true
```

BANG!

```
// a quick real example:

var x = 0;

if (!x) {
  // x must be FALSE to run this code
}
```

BANG!

```
// a quick real example:

var x = 0;

if (!x) {
  // x must be FALSE to run this code
  // or more accurately...
  // the OPPOSITE of x must be TRUE
  // because IFs are hungry for the TRUTH
}
```

BANG!

```
// another quick real example:

var likesCheetos = true;
var likesTakis = false;

if (likesCheetos && likesTakis) {

    alert(‘She likes Hot Cheetos AND Takis!’);

} else if (!likesCheetos || !likesTakis) {

    if (!likeCheetos) alert(‘She doesn’t like Hot Cheetos’);
    if (!likeTakis) alert(‘She doesn’t like Takis’);

}
```

NESTED IF STATEMENTS

NESTED IF STATEMENTS

```
// are perfectly fine
```

NESTED IF STATEMENTS

```
// are perfectly fine

if (likesCheetos && likesTakis) {

    alert(‘She likes Hot Cheetos AND Takis!’);

} else if (!likesCheetos || !likesTakis) {

    if (!likeCheetos) alert(‘She doesn’t like Hot Cheetos’);
    if (!likeTakis) alert(‘She doesn’t like Takis’);

}
```

NESTED IF STATEMENTS

```
// are perfectly fine

if (likesCheetos && likesTakis) {

    alert(‘She likes Hot Cheetos AND Takis!’);

} else if (!likesCheetos || !likesTakis) {
// this⤴⤴⤴⤴⤴⤴⤴⤴⤴⤴⤴⤴⤴⤴⤴⤴⤴⤴⤴⤴ is unnecessary

    if (!likeCheetos) alert(‘She doesn’t like Hot Cheetos’);
    if (!likeTakis) alert(‘She doesn’t like Takis’);

}
```

NESTED IF STATEMENTS

```
// are perfectly fine

if (likesCheetos && likesTakis) {

    alert(‘She likes Hot Cheetos AND Takis!’);

} else {

    if (!likeCheetos) alert(‘She doesn’t like Hot Cheetos’);
    if (!likeTakis) alert(‘She doesn’t like Takis’);

}
```

COMBINING OPERATORS

COMBINING OPERATORS

```
// magic
```

COMBINING OPERATORS

```
// a quick real example:

var age = 25;
var likesTakis = false;

if (age <= 13 && likesTakis) {

    alert(‘He probably likes Hot Cheetos too!’);

} else if (age===0 || age < 14 && !likesTakis || age > 29 && likesTakis) {

    alert(“He must be lying about his age.”);

}
```

COMBINING OPERATORS

```
// just don’t forget to be clear about your ordering...
```

```
} else if (age===0 || age < 14 && !likesTakis || age > 29 && likesTakis) {
```

COMBINING OPERATORS

```
// just don't forget to be clear about your ordering...



// most of the time, JS will do what you'd expect,
// but it never hurts to be explicit with parenthesis


} else if (age===0 || age < 14 && !likesTakis || age > 29 && likesTakis) {
```

COMBINING OPERATORS

```
// just don’t forget to be clear about your ordering...



// most of the time, JS will do what you’d expect,
// but it never hurts to be explicit with parenthesis


} else if (age===0 || (age<14 && !likesTakis) || (age>29 && likesTakis)) {
```

ORDER OF PRECEDENCE

```
// Parens ( ( ) )
// In/De-crements ( ++ -- )
// Bangs ( ! )
// Maths ( * / then + - )
// Comparisons ( < > <= >= )
// Equalities ( === !== )
// Logicals ( && || )
// Assignments ( = )


// see also:  http://bit.ly/MDNorder
```

OPERATORS, OPERATORS

```
// for more on the various types of operators:

// http://www.w3schools.com/js/js_comparisons.asp

// http://www.w3schools.com/js/js_operators.asp
```

BLOOMINGTON
CODE SCHOO

PREVIOUS CHALLENGE

```
// CHALLENGE

// Ask the user…
//  * What is your name?
//  * What is your age?

// Tell the user…
//  * their name and
//  * how old they will be at this time next year
```

FOR NEXT WEEK

```
// CHALLENGE

// modify the last challenge to do the following,
// in code, based on user’s age next year...

  - if the user will be under 15, ask their favorite color
  - if the user will be between 15 and 35, favorite food
  - if the user will be between 35 and 55, favorite book
  - if the user will be over 55, ask for __________
  - report back to the user with the data you collected
    (name, next age & favorite item)
```

ME

// brandonjp@gmail.com