JavaScript

Class 21

Notes on 224

- PHP has the `elseif` keyword; you should not use `else if`
- `else if` works with curly braces, but fails with colon-defined structures
- this is a syntax error:

```
<?php if ($a > $b): ?>
  <p>foo</p>
<?php else if ($a === $b): ?>
  <p>bar</p>
<?php endif; ?>
```

JS Quotes

- in PHP, single quotes denote a literal string, while double quoted strings perform variable interpolation
- in JS, there is no variable interpolation
- JS has no character data type; `'a'` is a string of length one
- it is syntactically legal use either single quotes or double quotes for a string
- the unbreakable style rule is to be consistent; don't mix the two forms
- stylistically, the strong favorite is to use only double quotes
- our style rule is double quotes

JS Regular Expressions

- we have looked at PHP regular expressions
- JS has all of the same power as PHP REs, but there are some differences

- there are two ways to create a JS RE
  1. a RE literal: `const re = /ab+c/;`
     the slashes are required; they cannot be a different character
     there are no quotes
  2. a RE object: `const re = new RegExp("ab+c");`
     the argument can be a literal string or a string variable
- note that if you need a backslash using 2, you have to escape it
  `const re = new RegExp("a\\*bc");` is identical to
  `const re = /a\*bc/;`
  both match any string containing the four characters a, *, b, and c, in that order

RE Methods

- a RE object has a number of methods; the most common is
  - test() looks for a match in a string; returns true or false
- a string object has a number of methods that use REs; the most common are:
  - search() looks for a match in a string; returns the index of the match, or -1 if there is no match
  - replace() looks for a match in a string, and returns a string in which the matched substring is replaced with the replacement substring

Testing RE Using Console

- look at file re.html

RE Examples

```
const string = "The quick brown fox";
const re = /\bB\w+/i;
console.log(re.test(string));
console.log(re.test("jumped over the lazy dog."));

const poem = "Roses are red, violets are blue";
const re = /\bB\w+/i;
console.log(poem.search(re)); /* 27 */

console.log(poem.replace(re, "cool"));
/* Roses are red, violets are cool */
```

Function Examples

- function.html + function.js

JSON Objects

- JS has objects (and now, classes)
- working with these is somewhat similar to other languages, but JS has some significant unique differences

- the simplest way to create an object in JS is a one-off object that does not belong to a class

```
const person =
{
  firstName: "Fred",
  lastName: "Flintstone"
}
```

- this is JSON, JavaScript Object Notation

JSON

```
const person =
{
  firstName: "Fred",
  lastName: "Flintstone",
  friends: ["Barney", "Betty"],
  toString: () => (this.firstName + " " + this.lastName)
  }
}
```

- fields are called properties
- separated by commas
- can add a new property later:
  `person.address = "123 Rocky Way";`
  now person has 5 properties, one of which is a function

JSON

- you will often see the field names in quotes also:

```
const person =
{
  "firstName": "Fred",
  "lastName": "Flintstone"
}
```

- JSON is the most common way to create objects in JS

Object Function

- you can write a function that returns an object
- this is not a class

```
function Person(fname, lname)
{
  this.firstName = fname;
  this.lastName = lname;
}

const fred = new Person("Fred", "Flintstone");

fred is now this object:
{
  firstName: "Fred",
  lastName: "Flintstone"
}
```

Object.create()

- you can use Object's static method create()
- this uses an existing object as a template to create another object (a clone) which can then be modified
- this is still not a class

```
const wilma = Object.create(fred);
wilma.firstName = "Wilma";

wilma is now this object:
{
  firstName: "Wilma",
  lastName: "Flintstone"
}
```

Class

- finally, as of ES6, we have actual classes

```
class PersonClass
{
  constructor(fname, lname)
  {
    this.firstName = fname;
    this.lastName = lname;
  }

  toString()
  {
    return this.firstName + " " + this.lastName;
  }
}

const barney = new PersonClass("Barney", "Rubble");
```

- Crockford doesn't think classes are a "good thing" so JSlint complains that all of this is undefined

Object.keys()

- in Java, C++, etc., every object belongs to a pre-defined class
- you know exactly what fields exist, because the class is a rigid template
- in JS, objects rarely belong to a class
- and, an object can be created with one set of fields, and gain other fields later
- so, how do you know what fields a given object has?

- Object.keys() returns an array of strings of the names of fields of an object

Object.keys()

```
const person =
{
  firstName: "Fred",
  lastName: "Flintstone",
  friends: ["Barney", "Betty"],
  toString: () => (this.firstName + " " + this.lastName)
  }
}
person.address = "123 Rocky Way";

Object.keys(person).forEach((item) => {console.log(item)});
```

- gives firstName lastName friends toString address