305CDE Lab 2

JavaScript Objects and Functions: Part I

October 2014

Overview

- Objects
- Functions

Objects

What Are Objects?

Example:

```
var employee = {
    firstName: "Colin",
    lastName: "Stephen",
    department: "Computing",
    hireDate: new Date()
};
```

JavaScript objects can be thought of as simple collections of *name-value pairs*. As such, they are similar to:

- Dictionaries in Python
- Hash tables in C and C++
- Associative arrays in PHP

Name-Value Pairs

The "name" part is a JavaScript string. Using "" signs is optional *if the string would be a valid JS variable name*:

- `"age"` OK, `age` OK
- `"first_name"` OK, `first_name` OK
- `"second-name"` OK, `second-name` NOT OK
- `"date of birth"` OK, `date of birth` NOT OK

The last two examples are strings that are not valid as JS variable names.

The "value" can be *any* JavaScript value:

- `112233`
- `"hello world"`
- `function() { \\ do something }`
- `false`

Object Creation

The preferred way to create objects in JS is using an "object literal":

```
var empty_object = {};

var physicist = {
  "first-name": "Albert",
  "second-name": "Einstein"
  "age": 135
};
```

Nested Objects

Remember that the value can be *any* JS value. That includes other objects. In other words: objects can be *nested*.

```
var flight = {
  airline: "BA",
  number: 882,
  departure: {
    IATA: "SYD",
    time: "2014-09-22 14:45",
    city: "Sydney"
  },
  arrival: {
    IATA: "LAX",
    time: "2014-09-23 10:32",
    city: "Los Angeles"
  }
};
```

Getting Object Values

Object values can be retrieved in two ways:

1. Use `[ ]` around a string with the name to retrieve as a suffix to the object name:

```
physicist["first-name"]  // returns "Albert"
flight["number"]  // returns 882
```

2. If the name is a legal JS name (and not a reserved word) then the `.` notation can also be used:

```
flight.airline  // returns 882
flight.departure.city  // returns "Sydney"
```

Undefined Values

If you try to retrieve a nonexistent name from an object, JS returns `undefined`:

```
physicist["middle-name"]  // returns undefined
flight.capacity  // returns undefined
```

TIP: the OR operator || can be used to fill in "default" values:

```
var middle = physicist["middle-name"] || "(none)"
var capacity = flight.capacity || "unknown capacity"
```

Undefined Objects

If you try to retrieve a value from an object that is `undefined`, JS throws a `TypeError` exception:

```
fakeObject["any-string"]  // throw "TypeError"

flight.capacity  // returns undefined
flight.capacity.minimum  // throw "TypeError"
```

TIP: the AND operator && can be used to guard against this problem:

```
flight.capacity  // undefined
flight.capacity.minimum  // throw "TypeError"
flight.capacity && flight.capacity.minimum  // undefined
```

Setting Object Values

Object values are set in two ways:

1. During object creation, unless your object is empty {}:

```
var employee = {name: "Colin"};
employee.name  // returns "Colin"
```

2. By assignment. This sets a new value if the name does not already exist. Otherwise, it updates the existing value:

```
physicist["middle-name"]  // returns undefined
physicist["middle-name"] = "Bob";
physicist["middle-name"]  // returns "Bob"

flight.arrival.city  // returns "Los Angeles"
flight.arrival.city = {full: "Los Angeles", short: "LA"}
flight.arrival.city.short  // returns "LA"
```

Call By Reference

Objects are passed around in JS programs "by reference". They are never copied.

```
var a = {};
var b = {};
a.test = "hello";
b.test  // returns undefined

var a = {};
var b = a;
a.test = "hello";
b.test  // returns "hello"

var stooge = {first: "Jerome", second: "Howard"}
var x = stooge;
x.nickname = "Curly";
var nick = stooge.nickname;
nick  // returns "Curly"
```