JavaScript
Functions and Objects

STEPHEN SCHAUB

Topics

- Functions
- Objects
- Arrays

One Thing to Rule Them All

- Functions are the key idea in JavaScript
- Create functions to
  - Abstract / reuse code
  - Define classes and methods
  - Create modules
- Functions are first class entities in JavaScript
  - Can be assigned to a variable
  - Can be passed to a function
  - Can be returned as a value from a function
  - Can be stored in an object or array

Function Basics

- Define with function statement

  -
    ```
    function add(num1, num2) {
      let result = num1 + num2;
      return result;
    }
    ```
- Invoke using the usual syntax:
  - `x = add(2, 2);`

Function Arguments

- JavaScript allows functions to be invoked with more / fewer arguments than specified in function definition
- For add() function on previous slide:
  - add(1, 2, 3, 4); // additional parameters ignored
  - add(1);   // num2 parameter has value **undefined**
- Additional parameters can be accessed through implicitly defined **arguments** array
- This allows for variable length arguments
  - It also allows for tricky bugs!

variable Length Arguments example

-
  ```
  function add(/* ... */) {
   let sum = 0;
   for (let i = 0; i < arguments.length; i++)
     sum += arguments[i];
   return sum;
  }

  let sum = add(1, 10, 100, 2, 3, 1000, 4, 5, 10000, 6);
  ```

Function Expressions

- The function definition
  - function add(num1, num2) { return num1 + num2; }
- … is equivalent to …
  - let add = function(num1, num2) { return num1 + num2; }
    or the equivalent lambda notation
  - let add = (num1, num2) => num1 + num2;
- Function expressions are **very useful**
  - Enable functional programming
  - Widely used in JavaScript frameworks

Functional Programming

- The Array.sort() method takes a comparison function
- Example:

```
function compareNumbers(a, b) { return a - b; }
numArray = [5, 2, 3];
numArray.sort(compareNumbers);
```

- Using a function expression:
  - numArray.sort((a, b) => a – b);

Function Gotcha

- Function names and variable names are in the same namespace
  - Because function names are really variable names bound to function objects

```
function add(num1, num2) { return num1 + num2; }

let sum = add; // create alternate name for add
let count = sum(2, 3); // invokes add

let add = 3; // replaces definition of add - OOPS!

add(2, 3); // an error; add is now a number, not a function
```

Nested Functions

- Unlike C/C++/Java, JavaScript allows functions to be nested

  -
    ```
    function foo(num1) {
      function boo() {
        return num1 + num2;
      }
      let num2 = 5;
      let result = boo();
      return result;
    }
    ```
- Inner functions ("closures") have access to all variables and parameters in their enclosing functions
  - Even after the enclosing function has returned

Closures are Powerful

- Enable:
  - Modules
  - Loop-less programming
  - Exotic functional capabilities
    - Currying
    - Partial application
    - (Useful in constructing frameworks)

Objects and Arrays

Objects

- An object is a map of name-value pairs
  - Basically, an associative array
- Create with an object literal expression:
  - let empty = {}; // empty object
  - let empty = new Object(); // empty object
  - let point = { x: 0, y: 0 };
- Properties can be added to an object through assignment
  - let obj = {};
    obj.x = 10; // add x property

Objects

- In object literal expressions, property names can be quoted or not
  - Quoting required if name is a reserved word
- Property values are expressions; quote only if a string

```
let person = {
  "name": "Fred Jones",
  "age": 15,
  "employed": false,
  "hireDate": new Date(2001,2,5)
}
```

```
let person = {
  name: "Fred Jones",
  age: 15,
  employed:  false,
  hireDate: new Date(2001,2,5)
}
```

Accessing Object Properties

- Two ways to access a property:
  - object.property
  - object["property"]
- Example:

  -
    ```
    let point = { x: 50, y: 100 };
    alert(point.x);        // 50
    alert(point["x"]);     // 50
    ```
- The array notation allows variables as indexes

  -
    ```
    index = "x";
    alert(point[index]); // 50
    ```
- Allows objects to double as associative arrays

Undefined Object Properties

- Recall that accessing an undeclared variable in an expression causes a runtime error
- Accessing an undefined object property in an expression results in the value **undefined**
  - let point = { x: 5, y: 10 };
    let z_coord = point.z;
    // point.z is undefined; z_coord is set to **undefined**

Iterating Object Properties

- Use a for-in loop:

  -
    ```
    for (let p in object) {
      alert(object[p]);
    }
    ```

Arrays

- Arrays are objects with a few additional methods
- Creating an array:

  -
    ```
    let arr = new Array();        // creates empty array
    let arr = [ ];                // creates empty array
    let arr = [ 1, 2, 3, 4, 5 ];  // creates array with 5 elements
    let arr = new Array(5);       // creates array with 5 slots (undefined)
    ```
- Arrays have a length property
  - arr.length - number of slots

Array Methods

- arr.push(*item*) - adds element to end of array
- arr.join(*separator*) - joins elements of array into a string
- arr.sort(*comparator*) - sorts array
- arr.forEach(*function*) - calls *function* on each member of arr
- arr.filter(*function*) - calls *function* on each member of arr and returns an array with selected elements

Resizing Arrays

- Arrays expand automatically as needed
  - let arr = [5, 10]; // arr.length == 2
    arr[5] = 10; // arr.length == 6; slots 0, 1, and 5 contain values
- Arrays can be truncated or expanded by assigning to length property
  - let arr = [1, 3, 5, 7, 9];
    arr.length = 3; // drops slots 3 and 4

Arrays vs. Objects

- Arrays and Objects look and smell a lot alike
  - You can add properties to arrays:
    - let arr = new Array();
      arr.color = "green"; // adds color property
  - You can add numbered slots to objects:
    - let obj = new Object();
      obj[1] = new Date();
  - What's the difference?
- Two important differences:
  - Objects do not have a length property
  - Objects do not have Array convenience methods