CS5220 Advanced Topics in Web Programming

Node.js Basics

Chengyu Sun
California State University, Los Angeles

About JavaScript

- Originally developed by Netscape
- Standardized as ECMAScript
- Many variations: *Client-side JavaScript, Node.js, TypeScript, JSX, …*

Year	Version
1997	1
2009	5 (ES5)
2015	6 (ES6 or ES 2015)
2016	7

Client-Side JavaScript

- JavaScript inside browser
- THE language for client-side web development
- Mostly ES6 except for older browser
- + Browser objects like `window` and `document`; DOM
- + HTML 5 APIs
- + jQuery

Node.js

- Standalone JavaScript
- Popular for server-side web development and used even for desktop applications
- Mostly ES6 (built on Chrome's V8 JavaScript engine)
+ Additional language features like a new runtime environment and modules (before ES6)
+ Many libraries

TypeScript

- JavaScript with Type
- Makes JavaScript more suitable for large software projects
  - Static type checking, improved tool support …
- ES6 and beyond (compiled to plain JavaScript)
- Used in Angular applications
- \+ Type and more

JSX

- JavaScript Extension for React
- Makes creating React elements in JavaScript a lot easier
- Based on ES6 (compiled to plain JavaScript)
- Used in React applications

Node.js Development Tools

- Node.js - https://nodejs.org/
  - npm package manager
  - Node Shell, a.k.a. Node REPL (Read-Eval-Print-Loop)
- Text editors for developers
  - Visual Studio Code by Microsoft
  - Atom by GitHub
  - Sublime Text

Basic Usage of NPM

- `npm init`
- `npm install/uninstall`
  - Local vs global
  - "devDependencies"
  - Install all dependencies
- `npm update`
  - Semantic versioning
- More on NPM

Elements of an Imperative Programming Language

- Comments
- Types
- Literals and Variables
- Operators and expressions
- Statements
- Functions
- Classes and Objects
- Packages

Elements of JavaScript

- Comments
- Types
- Literals and Variables
- Operators and expressions
- Statements
- Functions
- Classes and Objects
- Packages

The Strict Mode

- Put `"use strict";` (or `'use strict';`) before any statement in a script
- Change some default JavaScript behaviors to make it easier to spot errors, optimize performance, and migrate to future versions of ES
- See JavaScript Strict Mode for details

Comments

- Single-line comment: //
- Block comment: /* */

Types

- Boolean
- Number
- String
- Null
- Undefined
- (Symbol)
- Object

Primitive Types (i.e. types that define *immutable* values)

Literals

- Boolean: `true`, `false`
- Number: `123`, `4.56`
- String: `"hello"`, `'world'`
- Null and Undefined: `null`, `undefined`
- Template literal
- Object literal

Variables and Constants

```
let x;              // declare a variable x
x = 10;             // now x is a number
x = 'abc';                  // now x is a string
const y = 20;       // y is a constant
```

- JavaScript variables are dynamically typed (think of them as references instead of storage spaces)

Variable Scope

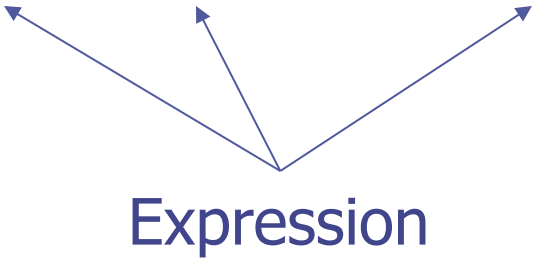
```
a = 10;          // global scope
var b = 20;      // function scope
let c = 30;      // block scope
const d = 40;    // block scope
```

- Scope example
  - Global vs function vs block
  - Strict vs non-strict mode
- Avoid using `var` or global variables

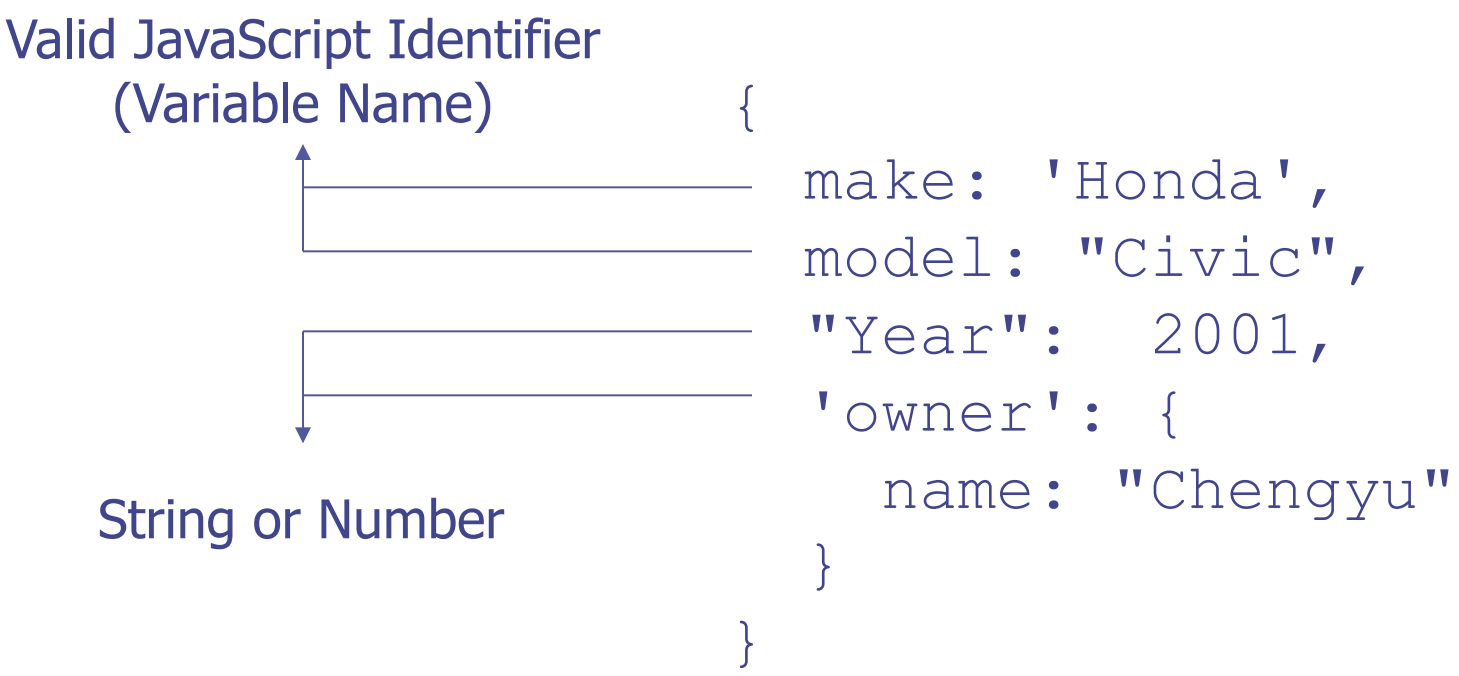
Template Literal

- A.K.A. Template String
- Example:

```
let a = 10;
let b = 20;

console.log( `${a}+${b} is ${a+b}` );
```



Object Literal



- An object literal consists of zero or more `key:value` pairs called *properties*

JSON (JavaScript Object Notation)

- Used as a data exchange format
- Based on a subset of JavaScript syntax
  - Strings are double quoted
  - Property keys are strings

```
{
  "make": "Honda",
  "model": "Civic",
  "year":  2001,
  "owner": {
    "name": "Chengyu"
  }
}
```

Object Property

Valid JavaScript Identifier
(Variable Name)

String or Number

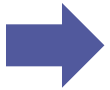
```
console.log( obj.make );

console.log( obj["make"] );

obj[0] = 10;
```

Properties can be dynamically added

- What if there's a variable `make`? ➔
  Computed Property

Property Value Shorthand

```
let name = 'Joe';
let user = {
  name: name,
  age: 20
}
```



```
let name = 'Joe';
let user = {
  name,
  age: 20
}
```

- Often used in object initializers, e.g.

```
function createUser(name, age) {
    return {name, age};
}
```

Array

```
a = ["x", "y", "z"];
a.b = "hello";
a[100] = 10;
```

```
{
  0: "x",
  1: "y",
  2: "z"
}
```

- An array is a special object where array elements are stored as object properties
  - An array may have “holes” (i.e. undefined elements)
- Array has built-in properties like `length`

Operators

- All Java operators, e.g. `+`, `-`, `=`, `&&` ...
- Strict equality/inequality: **`===`**, **`!==`**
  - `===` true if same type and same value
- Type operators: `typeof`, `instanceof`
- Object property operators: `in`, `delete`

Boolean and Equality …

0 == false	
"" == false	
0 == ""	
null == false	
undefined == false	
! null	
! undefined	
! obj	

… Boolean and Equality

- In JavaScript, a *truthy* value is a value that is considered true when encountered in a Boolean context. All values are truthy unless they are defined as *falsy* (i.e., except for `false`, `0`, `""`, `null`, `undefined`, and `NaN`).

Statements

- All common Java statements, e.g. `if`, `for`, `while`, `switch`, `break`, `continue` …
- `for…in` loop iterates over object property keys
- `for…of` loop iterates over object property values
- *Strict equality check* is used in `switch` statement
- Semicolon is optional but recommended

Functions as First-class Citizens

- In JavaScript, functions are actually objects
  - Assigned to a variable
    - Assigned as a property of an object
    - *Function literals* (a.k.a. *function expressions, anonymous functions*)
  - Passed as a function argument
  - Returned as a function result

Function Examples

```
function foo() {
  alert("foo");
}
```

Regular function declaration

```
bar = function() {
  alert("bar");
};
```

- Function literal
- Function assignment

```
setTimeout( bar, 5000 );
```

Function as parameter

```
setTimeout( function() {
  alert("foobar");},
  5000 )
```

Function literal as parameter

Function Arguments

```
function add(x,y) {
  return x+y;
}
```

```
add(10,20);
add("10","20");
add(10);
add(10,20,30);
```

- A special variable `arguments` hold all the arguments to a function
- `arguments` is not an array but similar to an array, e.g. `arguments.length`, `arguments[0]`, `arguments[1]`, …

Arrow Functions

- A.K.A. *lambda expressions, lambdas*
- A more concise way to write function literals
- Does not have its own `this` or `arguments` variables

```
function(a) {
  return a*2
}
```

```
(a)=>{return a*2}
```

```
a=>a*2
```

Lexical Scope and Closure ...

```
function foo(){
  let value=10;
  return () => {console.log(value++);}
}

var bar1 = foo();
var bar2 = foo();

bar1(); // ??
bar1(); // ??
bar2(); // ??
bar2(); // ??
```

… Lexical Scoping and Closure

- Inner function has access to the local variables of the outer function
- Lexical Scoping – the location in the source code where a variable is declared is used to determine where the variable is available
- Functions in JavaScript form *closures*.
  A closure is the combination of a function and the lexical environment within which that function was declared.

Function and Object

- Constructor
  - A function invoked with the `new` keyword
  - When invoked with `new`, the implicit parameter `this` is bound to a new object, and the function returns this object (unless there's an explicit return statement)
- Method
  - An object property that has a function value
  - `this` in a method refers to the object the method is called on

Example: Constructor

```
function Car( make, model, year )
{
    this.make = make;
    this.model = model;
    this.year = year;
}

var car1 = new Car("Honda", "Civic", 2016);
var car2 = new Car("Ford", "F-150", 2017);
```

Creating Objects

- Creating single object
  - Object literal
  - `new Object()` – more verbose than using object literal
- Creating objects using constructor
- Creating objects from an existing object using `Object.create()`

Example: Object.create()

```
var car3 = Object.create(car1);

console.log(car3.make);
console.log(car3.model);
console.log(car3);
```

- Prototype inheritance – an object inherits the properties of its *prototype*

Object Method Example

```
let user = {
  name: 'Joe',
  hello: function() {
    console.log(`My name is ${this.name}`);
  }
};
```

```
let user = {
  name: 'Joe',
  hello() {
    console.log(`My name is ${this.name}`);
  }
};
```

Getters and Setters: Accessor Properties

```
let user = {
  firstName: 'John',
  lastName: 'Doe',
  get name(){
    return this.firstName + ' '
      + this.lastName;
  },
  set name(value){
    [this.firstName, this.lastName] =
      value.split(' ')
  }
};
```

Destructuring assignment

Spread/Rest Syntax

```
let a = [1, 2, 3];

let b = [4, a, 5];
let c = [4, ...a, 5];
```

Spread

```
let [x, y, ...z] = [1, 2, 3, 4];
```

Rest

Error and Exception Handling ...

- The usage of `try`, `catch`, `finally`, `throw` are all similar to Java

```
try {
  // code...
} catch (err) {
  // error handling
}
```

… Error and Exception Handling

- The Error object
  - Properties: `name`, `message`, and (maybe) `stack`
- Try/catch doesn't quite work with asynchronous code

Some Important JavaScript Functions

- Array functions
  - Example: sum of array elements using `for` loop, `forEach`, and `reduce`
- JSON functions
- The lodash library

Readings

- The Modern JavaScript Tutorial
- Eloquent JavaScript by Marijn Haverbeke
- Exploring JS by Axel Rauschamyer

References

- MDN JavaScript Reference
- Node.js API Documentation