JavaScript: Introduction, Types

Lecture 19
History

- Developed by Netscape
  - "LiveScript", then renamed "JavaScript"
  - *Nothing* to do with Java!
- Interpreted
- Browser-based, client-side execution
- Standardized by ECMA ("ECMAScript")
  - But no one calls it that!
  - MIME type text/javascript
- Becoming popular outside of browsers
  - E.g. Node.js
- *c.f.*, JScript (MS), Dart (Google)…
Client-Side Execution

GET /news/index.php HTTP/1.1
Host: www.osu.edu
User-Agent: Mozilla/5.0 (X11; Ubuntu;...etc

<!DOCTYPE html>
<html lang="en">
  <head><title>My Page</title>
    <meta charset="utf-8" />
  </head>
  ...

Client-Side Execution

<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Something Short and Sweet</title>
    <meta charset="utf-8" />
  </head>
  <body>
    <p>Hello <a href="planet.html">World</a>!  
       <br />
    <img src="globe.png" alt="a globe"/>
  </p>
</body>
</html>
Client-Side Execution

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Something Short and Sweet</title>
    <meta charset="utf-8"/>
    <script>
      window.alert("Annoying!");
    </script>
  </head>
  <body>
    <p>
      Hello <a href="planet.html">World</a>!
    </p>
    <img src="globe.png" alt="a globe"/>
  </body>
</html>
```
Including Scripts

- Head: executed before body displays
  - Script (source) can be explicitly included
    
    ```html
    <script type="text/javascript">
      //default type in HTML 5!
      console.info("hi");
      ...
    </script>
    ```
  - Script can be linked in from external file
    
    ```html
    <script src="MyProgram.js"></script>
    ```
  - Recall: linking to CSS

- Inline: executed as body is displayed
- Browser blocks while downloading
  - Common advice: put scripts at end of body
  - Modern advice: use `<script src="..." async>`
Demo

☐ Simple "hello world"
  ■ HTML file containing JavaScript
  ■ Body is empty, script writes HTML output
  ■ Browser displays result

☐ Examining result
  ■ "view source": see JavaScript program
  ■ Firebug HTML tab: see rendered HTML

☐ Insert mistake
  ■ Replace writeln with println
  ■ Error console
Some Objects Provided Implicitly

- Some objects are created implicitly by the execution environment (browser)
- Document object (`document`)
  - `document.writeln()` puts output in body
- Window object (`window`)
  - Refers to browser's display window
  - Alert method pops up a dialogue `window.alert("Say \"cheese\"!");`
  - Prompt method pops up a dialogue `name = window.prompt("Enter name");`
Demo with Popups

- Alert window
- Prompt window
- Console output (info, warn, error)
Familiar (Java) Minor Syntax

- Statement separator `;`
  - Wrinkle: 's are optional!
    - Implicitly automatically inserted
    - But clearer and safer to include explicitly
- Statement blocks `{...}`
- Parentheses in expressions `(...)`
- Comments `//` and `/*...*/`
Familiar (Java) Operators

- **Arithmetic (numbers are floats)**
  - + - * / %
  - Wrinkles:
    - No diff in / between ints and floats!
    - % works on floats!

- **Relational**
  - < > <= >=
  - == !=
  - Wrinkle: === !==

- **Logical**
  - && || !
Familiar (Java) Statements

- **Assignment**
  - `= 
  - `+= -= *= /= %=`
  - `++ -- (pre and post)`

- **Conditionals**
  - `if (...)`, `if (...) ... else`
  - `switch (c)`
    - `case 'a': ... case 'b': ... default;`

- **Iteration**
  - `while (...)`, `do...while(...)`
  - `for (...;...;...)`
  - `break, continue`
Primitive vs Reference Types

- Distinction is similar to Java
- A variable is a "slot" in memory
- A variable can be primitive
  - The slot holds the value itself
  - Boolean, number, string, (null, undefined)
- A variable can be reference
  - The slot holds the pointer to the value
  - Arrays and objects (including functions!)
Primitive vs Reference Types

a
34.2

b
"hi"

c
4
0
-300
3.14

width: 12
height: 15
color: "blue"
Primitives: Checking Equality

```javascript
var a = 5;
var b = 5;
var c = 7;

if (a == b)  //=>true, slots are equal
if (a == c)  //=>false

var x = "hello";
var y = "hello";

if (x == y)  //=>true! c.f. Java
```
Primitives: Assignment is Copy

```javascript
var a = 5;
var b = a;  //copy contents of slot

b++;

if (a == 5)  //=>true, a is unchanged
```
Assignment is Copy (of Slot)
Primitives: Argument Passing

```javascript
function inc (param) {
    param++;  
}

var a = 5;
inc(a);  //copies contents of slot
if (a == 5)  //=>true
```
References: Equality/Assignment

```javascript
var a = {x:1, y:4}; // a new object
var b = {x:1, y:4}; // a new object

if (a == b) // => false slots unequal

a = b; // copy contents of slot

if (a == b) // => true
```
Assignment is Copy (of Slot)

\[
\begin{align*}
a &= b; \\
a &\neq b \\
a &= b
\end{align*}
\]
function inc (param) {
    param.x++;
}

var a = {x: 1, y: 4};
inc(a); //copy contents of slot
if (a.x == 2) //=>true
References: Argument Passing

```javascript
function inc (param) {
    param = {x: 2, y: 7};
}

var a = {x: 1, y: 4};
inc(a); //copies contents of slot
if (a.x == 2) //=>false
```
Wrinkle: == vs ===

- Recall + operator in Java:
  - Concatenation between strings
  - Addition between numbers
  - 3 + "4" also works! Results in "34"

- Similarly, == (!=) tries to make types match:
  - 3 == "3" is true

- To prevent implicit type conversion, use === (!==):
  - 3 === "3" is false
Demo: Iteration

- Table generated by Javascript
  - Prompt for initial value
  - Calculate interest series
  - Print out a row of table for each year
Static vs Dynamic Types

- **Static**: known at compile time
  - E.g. C, C++, Java, Ada
    ```
    int x
    char[] a
    FluffyCloud t
    void* d
    ```

- **Dynamic**: known only at run time
  - E.g. Python, PHP, Ruby, JavaScript
    ```
    var x
    var a
    var t
    var d
    ```
Static Types

- **a**: 34.2 (number)
- **b**: "hi" (string)
- **c**: num[]
- **d**: Shape
  - width: 12
  - height: 15
  - color: "blue"
Dynamic Types

a
34.2
var

b
"hi"
var

c
4
0
-300
3.14
var

[ ]

d
width: 12
height: 15
color: "blue"

Object
Function Signatures

- **Statically typed**
  ```java
  String parse(char[] s, int i) {... return e;}
  out = parse(t, x);
  ```
  - Parameter types (i.e. s and i) are declared
  - Return type (i.e. of parse) is declared
  - The compiler checks conformance of
    - (Declared) types of arguments (t, x)
    - (Declared) type of return expression (e)
    - (Declared) type of expression using parse (out)

- **Dynamically typed**
  ```java
  function parse(s, i) {... }
  out = parse(t, x)
  ```
  - You are on your own!
## Changing Types at Run-time

### Dynamic Types

```javascript
// a is undeclared
var a;

// a is undefined
a = "hi;"

// load-time error
a = "hi";

// a is number
a = 3;

// run-time error
a.crazy();
```

### Static Types

```javascript
// a is undefined
String a;

// a is null string
a = "hi;"

// compile-time error
a = "hi";

// compile-time error
int x = a;

// compile-time error
x.length();
```

### Static Types

```javascript
// a is undefined
String a;

// a is null string
a = "hi;"

// compile-time error
a = "hi";

// compile-time error
int x = a;

// compile-time error
x.length();
```
Summary

- Executes at client-side, in browser
  - Interpreted (not compiled)
- Basic syntax: operators, statements
- Objects: document, window...
- Types
  - Primitives: boolean, number, string, null, undefined
  - References: arrays, objects (& functions)
- Working with primitives and references
  - Checking equality
  - Assignment
  - Parameter passing
- Dynamic types (vs static types)