CS/COE 1520

pitt.edu/~ach54/cs1520

Client-side scripting: An introduction to Javascript

Why?

- By themselves, HTML and CSS can provide a description of the structure and presentation of a document to the browser
  - A *static* document
- According to the name of this class, we want to build *web applications*
  - We need to present a *dynamic* application to the user via the browser
    - … but why do we want to do this?
  - To do this, we'll need programs that can be fetched from the web and run within the browser

Scripting languages

- Programming languages designed for use within a given runtime environment
  - Often to automate tasks for the user
    - E.g.
      - bash, zsh, fish
      - Perl
      - Python
  - These languages are often *interpreted*
    - As opposed to being *compiled*

Compiled vs Interpreted

- Compiled: before being run, a program is compiled into machine code which is executed by the computer
  - E.g., C, C++, C#
- Interpreted: source code of a program is "executed" directly by an interpreter application
  - E.g., Python, Perl, Ruby, PHP
- Pretty simple, right?
  - What about Java?

Java doesn't fit this definition

Intermediate representations of code

- Java source code is compiled into bytecode
    - Which is then run by the JVM
        - … so is Java byte code an interpreted language?
- There are implementations for running both Python and Ruby on the JVM (Jython and JRuby)
- Both gcc and LLVM compile code in a series of phases:
    - Front-end compilers turn source code into an IR
    - IR is optimized
    - Optimized IR is turned into machine code
- Tools exist to run LLVM IR on the JVM

Javascript

- The de facto web client-side scripting language
- Javascript source code can be embedded within or referenced from HTML
  - Through the use of the `<script></script>` element
- It is an interpreted language
  - Javascript evaluated by the browser in rendering the HTML documents that contain/reference it
  - Javascript *engines* are the portion of the browser that interpret Javascript
    - Chrome has V8
    - Firefox has Spidermonkey

Javascript basics

- Variable names
  - Are case sensitive
  - Cannot contain keywords
  - Must begin with $, _, or a letter
    - Followed by any sequence of $'s, _'s, letters, or digits
- Numeric operators similar to those you know and love:
  - +, –, *, /, %, ++, --
- Comparison and boolean operators, too:
  - ==, !=, <, >, <=, >=, &&, ||, !
    - && and || are short circuited
- Strings
  - Have the + operator for concatenation
  - Have charAt, indexOf, toLowerCase, substring and many more methods
- Control statements similar to Java
  - if, while, do, for, switch
- Overall, it looks kind of like Java – intentionally

Javascript is dynamically typed

- Types are tied to values, not variables
- The types of the values stored in a given variable is determined at runtime
  - And can change over the run of the program!
  - This means that checks for type safety are evaluated at run time

Implications of dynamic typing in Javascript

- The + operator:
  - If one operand is a string value, the other will be coerced into a string and the two strings will be concatenated
- Numeric operators:
  - If one operand is a string value and it can be coerced to a number (e.g., "5"), it will be
  - If string is non-numeric, result is `NaN`
    - (NotaNumber)
  - We can also explicitly convert the string to a number using parseInt and parseFloat
- Comparisons:
  - == and != allow for type coercion
    - What does this mean?

Comparing both type and value

- An additional equality operator and inequality operator are defined to help deal with odd behavior presented by == and !=:
  - === returns true only if the variables have the same value and are of the same type
    - If type coercion is necessary to compare, returns false
  - !== returns true if the operands differ in value or in type

Functions

- `function foo(param1 , param2, param3) { … }`
- Return types are not specified
- Param types are not specified
- Functions execute when they are called, just as in any language
    - Because of this, function definitions should be in the head HTML element
    - E.g., `<head><script>function … </script></head>`
- Parameters are all passed by value
- No parameter type-checking
- Numbers of formal and actual parameters do not have to correspond
    - Extra actual parameters are ignored
    - Extra formal parameters are undefined
    - All actual parameters can be accessed regardless of formal parameters by using the arguments array

Javascript arrays

- More relaxed compared to Java arrays
  - Size can be changed and data can be mixed
  - Cannot use arbitrary keys
    - Similar to a hashmap
- Multiple ways to create arrays:
  - Using the new operator and a constructor with multiple arguments:
    - `var A = new Array("hello", 2, "you");`
  - Using the new operator and a constructor with a single numeric argument
    - `var B = new Array(50);`
  - Using square brackets to make a literal
    - `var C = ["we", "can", 50, "mix", 3.5, "types"];`

Javascript array length

- Like in Java, length is an attribute of all array objects
    - Unlike Java, this attribute is mutable
- In Javascript it does not necessarily represent the number of items or even memory locations in the array
    - Actual memory allocation is dynamic and occurs when necessary
    - An array with length = 1000 may in fact only have memory allocated for only 5 elements
- When accessed, empty elements are `undefined`

Some Javascript array methods

- concat
  - Concatenate two arrays into one
- join
  - Combine array items into a single string (commas between)
- push, pop, shift, unshift
  - Push and pop are a "right stack" (to/from end)
  - Shift and unshift are a "left stack" (to/from beginning)
- sort
  - Sort by default compares using alphabetical order
  - To sort numerically, we pass in a comparison function defining how the numbers will be compared
- reverse
  - Reverse the items in an array

Mutators!

Sorting comparison function pseudocode

```
function compare(a, b) {
   if (a is less than b by some ordering criterion) {
      return -1;
   }

   if (a is greater than b by the ordering criterion) {
      return 1;
   }

   // a must be equal to b
   return 0;
}
```

Javascript is an *object-based* language

- **NOT** object-oriented
  - It has and uses objects, but does not support some features necessary for object-oriented languages
    - E.g., Class inheritance and polymorphism are not supported

Javascript objects

- Javascript objects are represented as property-value pairs
  - In some ways similar to hashmaps
    - The object is analogous to the array backing the hashmap, and the properties are analogous to the keys
  - Property values can be data or functions (methods):

```
var my_tv = new Object();
my_tv.brand = "Samsung";
my_tv.size = 46;
my_tv.jacks = new Object();
my_tv.jacks.input = 5;
my_tv.jacks.output = 2;
```

Object details

- Note that the objects can be created and their properties can be changed dynamically
- Objects all have the same type: Object
  - Constructor functions for objects can be written, but these do not create new data types, just easy ways of uniformly initializing objects

```
function TV(brand, size, injacks, outjacks) {
    this.brand = brand;
    this.size = size;
    this.jacks = new Object();
    this.jacks.input = injacks;
    this.jacks.output = outjacks;
}
…
var my_tv = new TV("Samsung", 46, 5, 2);
```