Week 2 Overview

- Week 1 review
  - HTML documents
    - Document Type Definition
    - Elements (tags)
    - Attributes
    - Entities
  - Inline and external JavaScript
Week 2 Overview

• Outcomes
  • Describe the inputs, activities, and outputs of each step in the software development life cycle.
  • Describe arithmetic, relational, and logical operators in terms of their input and output data types.

• Outcomes
  • Declare, define, and use variables in a script.
Software Lifecycle

• Outcome
  • Describe the inputs, activities, and outputs of each step in the software development life cycle.
Software Lifecycle

Documents

Waterfall Process

Code

Production code

Software Lifecycle

Documents

Requirements

Analysis and Design

Implementation

Testing

Maintenance

www.franklin.edu
Software Lifecycle

• Requirements
  • Functional
    • What the software should do
    • e.g. “on input X produce Y…”
  • Non-functional (qualitative)
    • Criteria against which the system is measured
    • e.g. “…within 2 seconds or less”

Software Lifecycle

• Analysis and Design
  • Determine architecture
    • System
    • Software
  • Determine what is done in software
  • Create abstract models
  • High-level and low-level
  • Generally, “what” not “how”
Software Lifecycle

• Implementation
  • Translate design into code
    • Algorithms
    • Objects
    • Functions
    • Control structures
  • i.e. what is generally considered to be “programming”

• Testing
  • Validates code two ways
    • That it does what it should do
      • Functional requirements
      • Non-functional requirements
    • That it doesn’t do what it shouldn’t do
      • Graceful failure
      • Error recovery
Software Lifecycle

- Maintenance
  - Ongoing development to
    - Fix bugs
    - Add new features
  - Can be more than 80% of man-hours
  - Typically what separates “academic” projects from “production” projects!

Software Lifecycle

- Waterfall advantages
  - Simplicity
  - Easy to benchmark
  - Clearly delineated milestones
Software Lifecycle

- Waterfall problems
- Assumptions
  - Stable requirements
  - Stable technologies/staffing
  - Early risk identification
  - Familiarity with the problem domain
  - No need for feedback in the system
- Result: early mistakes are costly

Spiral Process
Software Lifecycle

- Advantages
  - Changing requirements
  - Early problem discovery
  - Always working software
  - Extensive feedback

- Problems
  - Architecture suffers
Variables and Data Types

- **Outcome**
  - Declare, define, and use variables in a script.
Variables and Data Types

• **Boxes**
  • In the real world:
    • Some boxes are empty
    • Some boxes hold things
    • Box contents can be replaced
    • Boxes can hold more than one thing

Variables and Data Types

• **Variables**
  • In programming:
    • Some variables are “empty”
    • Some variables hold one item
    • Variable contents can be replaced
    • Variables can hold more than one thing
      (an array)
Variables and Data Types

- Variables have 4 key properties
  - Have a *name*
  - Have a *value*
  - Have a *data type*
  - Have a *scope*
- Can have operations performed on them

Declaring a variable

```javascript
var myVariable;
```

**var**: keyword to create a “box” to hold data.

**myVariable**: an *identifier*. The name of the variable being created. You invent your own descriptive name for variables.
Variables and Data Types

- Declaring a variable

\[\text{var myVariable;}\]

- Declaring multiple variables

\[\text{var myVariable, yetAnotherVariable;}\]

- Defining a variable – an initial value

\[\text{var myVariable; myVariable = 5;}\]

The variable name is an l-value (something that can appear on the left hand side of an assignment statement)
Variables and Data Types

• Defining a variable – an initial value

```javascript
var myVariable;
myVariable = 5;
```

Assignment operator: puts the contents on the right hand side into the “box” on the left hand side.

5 is an r-value (something that can appear on the right hand side of an assignment statement).
Variables and Data Types

• Defining a variable – an initial value

```javascript
var myVariable;  
myVariable = 5;
```

• Doing both at once

```javascript
var myVariable = 5;
```

• Getting user input into variables

```javascript
var firstName = prompt("Enter your first name");  
var age = prompt("Enter your age");
```

`prompt`: a method of the `window` object that opens an input dialog box with the string parameter as a visual queue. Always returns a string.
Variables and Data Types

• Rules for variable names (identifiers)
  • Cannot be a reserved word (Gosselin, p. 60)
  • Must start with [A-Z, a-z, _, $]
  • Subsequent characters can also include [0-9]
  • No spaces allowed

Example

<table>
<thead>
<tr>
<th>Variable</th>
<th>Valid or invalid?</th>
<th>If invalid, why?</th>
</tr>
</thead>
<tbody>
<tr>
<td>aSampleID</td>
<td></td>
<td></td>
</tr>
<tr>
<td>First_Name</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1forTheMoney</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$big&amp;tall</td>
<td></td>
<td></td>
</tr>
<tr>
<td>document</td>
<td></td>
<td></td>
</tr>
<tr>
<td>class</td>
<td></td>
<td></td>
</tr>
<tr>
<td>my age</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Variables and Data Types

- Informal rules for variable names
  - Should not conflict with another built-in identifier.
  - Should use camelCaseConventions
  - Should be descriptive of their purpose
    - Exceptions: i, j, k, etc., used as counting loop variables

Variables and Data Types

- Data types
  - Each variable has a *type* that determines which operations can be performed on it.
  - e.g. numbers can have arithmetic performed on them, strings can be concatenated, etc.
Variables and Data Types

• Data types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integers</td>
<td>42</td>
<td>A whole number -2^31 through 2^31</td>
</tr>
<tr>
<td>Reals</td>
<td>6.023E23</td>
<td>A number with a decimal point</td>
</tr>
<tr>
<td>Boolean</td>
<td>true</td>
<td>Either true or false</td>
</tr>
<tr>
<td>String</td>
<td>&quot;lorem ipsum&quot;</td>
<td>A sequence of character data</td>
</tr>
<tr>
<td>Undefined</td>
<td>null</td>
<td>Declared but uninitialized variable</td>
</tr>
<tr>
<td>Null</td>
<td>null</td>
<td>The &quot;empty&quot; object</td>
</tr>
<tr>
<td>Object</td>
<td>new Date()</td>
<td>Any user defined object</td>
</tr>
</tbody>
</table>

var lastName = "Smith";
var numDependents = 3;
var dateOfBirth = new Date(1973, 11, 29);
var canVote = true;
document.writeln(typeof lastName);  // string
document.writeln(typeof numDependents);  // number
document.writeln(typeof dateOfBirth);  // object
document.writeln(typeof canVote);  // boolean
document.writeln(typeof (typeof 42));  // string
Variables and Data Types

• Scope
  • A range of lines during which the variable is “live.”
    • **Static scoping**: lifespan of a variable can be determined by inspecting the source code.
    • **Dynamic scoping**: lifespan of a variable can only be determined as the program is running.

• JavaScript is (on the whole) *statically* scoped.
  • **Global scope**: any variable created (declared) outside of a function or without the `var` keyword.
  • **Local scope**: any variable created within a function and using the `var` keyword.
Operators

Outcome

- Describe arithmetic, relational, and logical operators in terms of their input and output data types.
JavaScript Operators

- Data types determine valid operators
- Can add, subtract, multiply, and divide numbers but not Booleans
- Can compare numbers and strings but not objects.
- Can use *and*, *or*, and *not* on Booleans, but not strings

JavaScript Operators

- Arithmetic operators – *math*

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Adds two numbers yielding their sum</td>
</tr>
<tr>
<td>- (binary)</td>
<td>Subtracts two numbers yielding their difference</td>
</tr>
<tr>
<td>*</td>
<td>Multiplies two numbers yielding their product</td>
</tr>
<tr>
<td>/</td>
<td>Divides two numbers yielding their quotient</td>
</tr>
<tr>
<td>%</td>
<td>Divides two numbers yielding their remainder</td>
</tr>
<tr>
<td>- (unary)</td>
<td>Negates a single number</td>
</tr>
</tbody>
</table>
JavaScript Operators

• Operator precedence
  • Just like in math

```
var x = 4 + 2 * 3 - 1; // x has value 9
var y = 4 + 2 * (3 - 1); // y has value 8
```

• Complete table Gosselin p. 95-96

JavaScript Operators

• Relational Operators – comparison

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal to</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal to</td>
</tr>
<tr>
<td>==</td>
<td>Equal to</td>
</tr>
<tr>
<td>!=</td>
<td>Not equal to</td>
</tr>
<tr>
<td>===</td>
<td>Equal to and of same type</td>
</tr>
<tr>
<td>!==</td>
<td>Not equal to or not of same type</td>
</tr>
</tbody>
</table>
JavaScript Operators

• Relational Operators – comparison

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$3 &lt; 21$</td>
<td></td>
</tr>
<tr>
<td>&quot;Fred&quot; &lt;= &quot;Ginger&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;3&quot; &lt; &quot;21&quot;</td>
<td></td>
</tr>
<tr>
<td>$3 &gt;= 3$</td>
<td></td>
</tr>
<tr>
<td>&quot;3&quot; == 3</td>
<td></td>
</tr>
<tr>
<td>&quot;3&quot; !== 3</td>
<td></td>
</tr>
<tr>
<td>&quot;3&quot; !== 3</td>
<td></td>
</tr>
</tbody>
</table>

JavaScript Operators

• Relational Operators – comparison

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$3 &lt; 21$</td>
<td>true</td>
</tr>
<tr>
<td>&quot;Fred&quot; &lt;= &quot;Ginger&quot;</td>
<td>true</td>
</tr>
<tr>
<td>&quot;3&quot; &lt; &quot;21&quot;</td>
<td>false</td>
</tr>
<tr>
<td>$3 &gt;= 3$</td>
<td>true</td>
</tr>
<tr>
<td>&quot;3&quot; == 3</td>
<td>true</td>
</tr>
<tr>
<td>&quot;3&quot; === 3</td>
<td>false</td>
</tr>
<tr>
<td>&quot;3&quot; !== 3</td>
<td>true</td>
</tr>
</tbody>
</table>
### JavaScript Operators

#### Logical Operators – join Booleans

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;&amp;</td>
<td>Logical AND</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>!</td>
<td>Logical NOT</td>
</tr>
</tbody>
</table>

#### Expression Value Table

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td></td>
</tr>
<tr>
<td>true &amp;&amp; !false</td>
<td></td>
</tr>
<tr>
<td>true</td>
<td></td>
</tr>
<tr>
<td>true</td>
<td></td>
</tr>
<tr>
<td>true &amp;&amp; false</td>
<td></td>
</tr>
<tr>
<td>!(true</td>
<td></td>
</tr>
</tbody>
</table>

- Complete table – Gosselin p. 95-96
JavaScript Operators

• Logical Operators – join Booleans

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td></td>
</tr>
<tr>
<td>true &amp;&amp; !false</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td></td>
</tr>
<tr>
<td>true</td>
<td></td>
</tr>
<tr>
<td>true &amp;&amp; false</td>
<td></td>
</tr>
<tr>
<td>!(true</td>
<td></td>
</tr>
</tbody>
</table>

• Complete table – Gosselin p. 95-96

JavaScript Operators

• Compound assignment

<table>
<thead>
<tr>
<th>Operator</th>
<th>Shortcut for</th>
</tr>
</thead>
<tbody>
<tr>
<td>left += right</td>
<td>left = left + right</td>
</tr>
<tr>
<td>left -= right</td>
<td>left = left - right</td>
</tr>
<tr>
<td>left *= right</td>
<td>left = left * right</td>
</tr>
<tr>
<td>left /= right</td>
<td>left = left / right</td>
</tr>
<tr>
<td>left %= right</td>
<td>left = left % right</td>
</tr>
</tbody>
</table>
JavaScript Operators

• Increment (++) and decrement (--)
  • Shortcut for adding 1 to a variable
  • Pre- versus post- operators
    • Pre- : ++ or -- operation on variable first, then yield the variable value
    • Post- : yield the variable value, ++ or -- operation on the variable last

Assume x is 10 initially

<table>
<thead>
<tr>
<th>Example</th>
<th>New value of y</th>
<th>New value of x</th>
</tr>
</thead>
<tbody>
<tr>
<td>y = x++</td>
<td></td>
<td></td>
</tr>
<tr>
<td>y = ++x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>y = x--</td>
<td></td>
<td></td>
</tr>
<tr>
<td>y = --x</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
JavaScript Operators

- Increment (++) and decrement (--)
  - Assume x is 10 initially

<table>
<thead>
<tr>
<th>Example</th>
<th>New value of y</th>
<th>New value of x</th>
</tr>
</thead>
<tbody>
<tr>
<td>y = x++</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>y = ++x</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>y = x--</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>y = --x</td>
<td>9</td>
<td>9</td>
</tr>
</tbody>
</table>

JavaScript Operators

- Conditional operator
  - Syntax:
    `<boolean_expression> ? <true_part> : <false_part>`
  - Similar to Excel IF function

```javascript
var number = prompt("Enter an integer");
document.writeln("The number was " +
    (number % 2 == 0 ? "even" : "odd"))
```
Questions?
Self Quiz

• What are the rules about how variables can be named?
• Which kind of operators combine Boolean expressions to create a Boolean result?
• Which kind of operators combine numbers to create a Boolean result?

Self Quiz

• What kinds of operators combine numbers to make a number result?
• What are the stages of the software development lifecycle?
• What do you do in each stage?
• What is the output of each stage?
• Which stage takes the most time?
Self Quiz

- How are the spiral-model and the waterfall-model of software development similar? Different?
- What is a l-value? R-value?
Upcoming Deadlines

- Pre-class exercise 3 – Due Jan 19
- Homework 2 – Due Jan 19