JavaScript

Lecture 17

CS 638 Web Programming

Overview of lecture

- On client side programming with JavaScript
- The core language
- Arrays
- Objects
- Variables, typing, and scoping

Why is JavaScript important?

- Web pages can contain JavaScript programs executed inside the browser
  - Supported by all major browsers
    - Microsoft’s version called Jscript (the language is the same)
    - User may disable JavaScript due to security fears
      - This is default for some newer versions of Internet Explorer
- Client-side programming important for web because
  - Can promptly validate user input
  - Can update the web page without postback to server
  - Allows page to react to user actions other than pushing a "submit" button – more interactivity
What is JavaScript?

- Interpreted, object-oriented programming language with dynamic typing
- Introduced by Netscape with Netscape 2.0 in 1995
- Standardized as ECMAScript by ECMA (European Computer Manufacturers Association)
- Not related to Java other than the name
- Tightly integrated with browser
  - Can handle many types of events generated by the normal interaction between user and browser
  - Can modify the internal objects based on which the browser renders the web page

Overview of lecture

- On client side programming with JavaScript
- The core language
- Arrays
- Objects
- Variables, typing, and scoping
Syntax similar to C++

- Statement block delimited by “{” and “}”
- Statements separated by ‘;’
- If single command per line, the ‘;’ can be left out
- Spaces, indentation have no semantic value
- Comments after ‘//’ or between ‘/*’ and ‘*/’
- Identifiers contain $, _, letters and digits
- Cannot start with a digit
- Variable names and keywords case sensitive

The if and switch statements

```c
if(expression)
    statement
else if (expression1)
    statement1
else if (expression2)
    statement2
... else if (expressionk)
    statementk
else
    catch_all_statement
```

```c
switch(n){
    case 1: // if n==1
        statements1
    break;
    case 2: // if n==2
        statements2
    break;
    default: // otherwise
        statements3
}
```

Loops

```c
while (expression)
    statement
```
```
do
    statement
while (expression);
```
```
for (init;test;incr)
    statement
```

- break ends the innermost loop or switch
- continue jumps to the end of loop
- Labels can be used with nested loops
- Labels can be used with nested loops
- `outer: for(i=0;i<5;i++){
    for(j=0;j<5;j++){
        a[i][j]=a[i][j]+1;
    }
    break outer;
}"
Functions and exceptions

- Defined as
  
  ```javascript
  function fname(args) {
    statements;
  }
  ```

- Invoked as
  
  ```javascript
  fname(args)
  ```

- Execution of function can be terminated with
  
  ```javascript
  return expression;
  ```

- To throw an exception
  
  ```javascript
  throw expression;
  ```

- To catch an exception
  
  ```javascript
  try{
    statements1
  }catch(e){
    statements2
  }finally{
    statements3
  }
  ```

Overview of lecture

- On client side programming with JavaScript
- The core language
- Arrays
- Objects
- Variables, typing, and scoping
JavaScript Arrays

- Initialized by assigning an array literal `a=[]` or by using constructor `a=new Array(1,2,3)`
- Elements’ indices start with 0
- Can grow by assignment to inexistent element
- The `length` property gives length – `a.length`
- Arrays are sparse: `a=['first']
  `a[99]='hundredth' gives a two elements and a length of 100`
- Special value `undefined` returned when inexistent element read – no exception thrown!

Useful array methods

- `join()` combines elements into a single string
- Default separator `','` but can specify different one
- `String.split()` method gives array of substrings
- `reverse()` reverses the order of elements
- `sort()` sorts elements in alphabetic order
- `slice(from,to)` returns elements between indices
- `from` and `to`
  - Negative “indices” give distance from end of array
- `pop()` deletes and returns last element
- `push(e)` adds `e` to end of array, returns new length
- `shift()` deletes and returns first element (shifts others)
- `unshift(e)` adds `e` to beginning of array (shifts others)
Overview of lecture

- On client side programming with JavaScript
- The core language
- Arrays
- Objects
- Variables, typing, and scoping

JavaScript objects

- Objects are little more than associative arrays
  - `obj.property` is same as `obj["property"]`
  - To create a new property, just assign to it
  - Object methods are properties whose values are functions
    - They can use the keyword `this` to refer to object
- Prototype-based inheritance (no classes)
  - Object `x` also inherits all properties of `x.prototype`
  - The constructor is just a function that uses the keyword `this` to access the new object
  - "Class properties" simulated with properties of constructor

C# vs. JavaScript objects

```csharp
using System;

public struct Circle
{
    public double x, y, radius;

    public Circle(double x, double y, double radius)
    {
        this.x = x;
        this.y = y;
        this.radius = radius;
    }

    public double getArea()
    {
        return radius * radius * Math.PI;
    }

    public double getDist()
    {
        return Math.Sqrt((x * x + y * y));
    }
}

class Program
{
    static void Main(string[] args)
    {
        Circle c = new Circle(2, 4, 3);
        Console.WriteLine("Area is {0}", c.getArea());
        Console.WriteLine("Distance is {0}", c.getDist());
    }
}
```

```javascript
var Circle = function(x, y, r)
{
    this.x = x;
    this.y = y;
    this.radius = r;
}
Circle.prototype.getArea = function()
{
    return this.radius * this.radius * Circle.pi;
}
Circle.prototype.getDist = function()
{
    return Math.sqrt(this.x * this.x + this.y * this.y);
}
Circle.pi = 3.14;
c = new Circle(2, 4, 3);
document.write("Area is " + c.getArea() + "<br/>");
document.write("Distance is " + c.getDist() + "<br/>");
```
Traversing an object

**The for/in loop**

```javascript
point={x:2,y:4};
for (p in point)
    document.write(p);
```

- Does not go through inherited properties
- For arrays, the `for/in` loop goes through the indices of elements

**The in operator**

```javascript
point={x:2,y:4};
has_x="x" in point;
has_z="z" in point;
```

- Checks inherited properties
- For arrays, the `in` operator tests whether the element with the given index exists

Overview of lecture

- On client side programming with JavaScript
- The core language
- Arrays
- Objects
- Variables, typing, and scoping

Most important types

- **Numbers** – 15 or `0xff` or 5.19 or 5.7e15
- **Strings** – ‘abc’ or "abc" or ‘a’\'\nb’
- Concatenation using `+`
- **Booleans** – `true` or `false`
- **Functions** – `sq=function(x){return x*x;}`
- **Arrays** – `[1,2,3]` or `[‘abc’,[2,5],8]`
- **Objects** – `{x:2,y:3}` or `{color:”blue”,age:5}`
- No classes – all objects are of the same type
- Functions and arrays are also objects
### JavaScript variables

- All variables untyped – `x=4; x='xyz'; x=[]`
- Variables are either global or local to function
  - No block scope!!!
- Variables are declared using the `var` keyword
- Assigning to unused name implicitly declares it as a global
- Reading an undeclared variable causes an error

<table>
<thead>
<tr>
<th>Types</th>
<th>Passed/copied by</th>
<th>Comparison</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean, number</td>
<td>value</td>
<td>value</td>
</tr>
<tr>
<td>string</td>
<td>immutable</td>
<td>value</td>
</tr>
<tr>
<td>object (arrays, functions)</td>
<td>reference</td>
<td>reference</td>
</tr>
</tbody>
</table>

### Some JavaScript operators

<table>
<thead>
<tr>
<th>Operator(s)</th>
<th>Operand type(s)</th>
<th>Operation performed</th>
</tr>
</thead>
<tbody>
<tr>
<td>[]</td>
<td>object, identifier</td>
<td>Property access</td>
</tr>
<tr>
<td>[]</td>
<td>array, integer</td>
<td>Array access</td>
</tr>
<tr>
<td>[]</td>
<td>function, argument</td>
<td>Function call</td>
</tr>
<tr>
<td>++,--</td>
<td>value</td>
<td>Increment, decrement</td>
</tr>
<tr>
<td>.</td>
<td>object, identifier</td>
<td>Property access</td>
</tr>
<tr>
<td>[]</td>
<td>array, integer</td>
<td>Array access</td>
</tr>
<tr>
<td>+,</td>
<td>number, number</td>
<td>Multiplication, division, remainder</td>
</tr>
<tr>
<td>-</td>
<td>number, number</td>
<td>Addition, subtraction</td>
</tr>
<tr>
<td>&lt;,&gt;,&gt;&gt;</td>
<td>integer, integer</td>
<td>Shift operations</td>
</tr>
<tr>
<td>==,!=</td>
<td>any, any</td>
<td>Equality, inequality</td>
</tr>
</tbody>
</table>

- Silbase operators: boolean operators, conditional operator
- =, +=, -= | value, any | Assignment |
- %=, -=, *= | value, any | Assignment with operation |