JavaScript

Overview

- Dynamic and Weak Typing
- Java-based syntax
- Multi-paradigm support
  - imperative and object-oriented
  - functional
- Object-oriented programming using prototypes
- Programming constructs
  - **Selection statements**: if/else, switch
  - **Looping**: while, for, for...in
  - **Modularization**: function, Object
  - **Built-in Types**: String, Number, Boolean, Array, Function, Object

Dynamic Typing

- a=10 → number
- a='10' → string

- Type is determined at the runtime
- Use the rvalue to determine the type of lvalue

Weak Typing

- '10' == 10 → true or false?
- alert('10' + 10) → ?
- alert('10' * 10) → ?
- alert('10' * '10') → ?

- Operations may not be type-safe i.e. type rules not enforced properly
- Implicit type conversion
- Type rules aren't uniform - unpredictable behavior

Conditionals

- **if / else**

```
...
if (marks > 50) {
  alert('pass');
}
else {
  alert('fail')';
}
```

Conditionals

- **Switch**

```
…
switch(grade){
  case 'A':
  case 'B':
  case 'C':
  case 'D':
     alert('pass');
     break;
  case 'F':
     alert('fail');
     break;
  default:
     alert('incomplete');
}
```

Loops

- **for loop**

```
var nums='';
for(i=0; i < 10; i++){
   nums = nums + i;
}
alert(nums);
```

Loops

- **while loop**

```
var nums='';
var i=0;
while( i < 10 ){
   nums = nums + i + '\n';
   i++;
}
alert(nums);
```

Loops

- **do/while loop**

```
var nums='';
var i=0;
do {
   nums = nums + i + '\n';
   i++;
} while( i < 10 );
alert(nums);
```

Functions

- The basic unit of modularization in JavaScript
- Functions serve multiple purposes in JavaScript
- Can be treated just like a type
- Basic Syntax

```
function name(var1,var2,var3,...){
  return;
}
```

Functions

- Some examples

```
function product (a,b) {
  return a*b;
} // returns product

var product = function (a,b) {
   return a*b;
} // same function as above – declared differently
```

Functions

- Yet another example

```
var product = function () {
   return arguments[0] * arguments[1];
}
alert ( product (10, 20)) → 200
```

- Number of parameters defined may differ from number of parameters passed
- Each function has an implict parameter named **arguments** – an array of arguments passed at run-time

Functions

- Scoping rules
  - All variables declared inside function with **var** keyword have function scope – i.e. they are local variables

```
var a=1;
alert(a);  // output 1

function scopeTest(){
  var a=2;
  alert(a);
}

scopeTest(); // output 2
alert(a); // output 1
```

```
var a=1;
alert(a);  // output 1

function scopeTest(){
  a=2;
  alert(a);
}

scopeTest(); // output 2
alert(a); // output 2
```

Quiz

- Write a function that takes as an input a number and prints it in reverse order
    - e.g. 123 printed as 321

- Write a function that calculates average of numbers passed as parameter

- Write a recursive function to calculate factorial
    - e.g. 5 ! = 120

Objects

- An object is a named collection of properties and methods
- Objects in JavaScript need not belong to a class
- Objects can be instantiated
  - Using literal notation; or
  - Directly from default Object:

    var obj = new Object();

    // simplest way to get started with objects in javascript

Objects

```
var item = {
  title : "HTML Specification",
  status : "available",
  isAvailable : function (){
       return this.status == "available";
  }
}
```

```
alert(item.title); // HTML Specification
alert(item.isAvailable()); // true
```

Objects

```
var item = new Object();
item.title = "HTML Specification";
item.status = "available";
item.isAvailable = function (){
        return this.status == "available";
}
```

```
alert(item.title); // HTML Specification
alert(item.isAvailable()); // true
```

Using Object Constructors

```
function Item(t){

  this.title = t;
  this.status = "available";
  this.isAvailable = function (){
       return this.status == "available";
  }

}
```

```
var item1 = new Item("HTML Specification");
alert(item1.title); // HTML Specification
alert(item1.isAvailable()); // true
```

```
var item2 = new Item("Javascript Specification");
alert(item2.title); // Javascript Specification
alert(item2.isAvailable()); // true
```

Iterating Object Properties

```
var obj = new Item("HTML Specification");
var output = '';

for(x in obj){
      output += x + ": " + obj[x] + ";" ;
}

alert(output);
```

Extending Objects

```
var item1 = new Item("HTML Specification");
alert(item1.title); // HTML Specification
alert(item1.isAvailable()); // true

item1.issue = function(){
  this.status = "issued";
}

alert( item1.isAvailable() ); // true

item1.issue();

alert( item1.isAvailable() ); // false
```

```
var item2 = new Item("Javascript Specification");
alert(item2.title); // Javascript Specification
alert(item2.isAvailable()); // true





alert( item2.isAvailable() ); // true

item2.issue();                    // Exception

alert( item2.isAvailable() ); // true
```

Extending all objects of a type Using prototypes

```
var item1 = new Item("HTML Specification");
alert(item1.title); // HTML Specification
alert(item1.isAvailable()); // true

Item.prototype.issue = function(){
  this.status = "issued";
}

alert( item1.isAvailable() ); // true

item1.issue();

alert( item1.isAvailable() ); // false
```

```
var item2 = new Item("Javascript Specification");
alert(item2.title); // Javascript Specification
alert(item2.isAvailable()); // true





alert( item2.isAvailable() ); // true

item2.issue();

alert( item2.isAvailable() ); // false
```

Extending all objects of a type Using prototypes

```
var item1 = new Item("HTML Specification");
alert(item1.title); // HTML Specification
alert(item1.isAvailable()); // true

item1.__proto__.issue = function(){
  this.status = "issued";
}

alert( item1.isAvailable() ); // true

item1.issue();

alert( item1.isAvailable() ); // false
```

```
var item2 = new Item("Javascript Specification");
alert(item2.title); // Javascript Specification
alert(item2.isAvailable()); // true




alert( item2.isAvailable() ); // true

item2.issue();

alert( item2.isAvailable() ); // false
```

Extending all objects of a type Using prototypes

```
var item1 = new Item("HTML Specification");
alert(item1.title); // HTML Specification
alert(item1.isAvailable()); // true

Object.getPrototypeOf(item1).issue = function(){
  this.status = "issued";
}

alert( item1.isAvailable() ); // true

item1.issue();

alert( item1.isAvailable() ); // false
```

```
var item2 = new Item("Javascript Specification");
alert(item2.title); // Javascript Specification
alert(item2.isAvailable()); // true





alert( item2.isAvailable() ); // true

item2.issue();

alert( item2.isAvailable() ); // false
```

Constructors – recommended approach

```
function Item(t){

  this.title = t;
  this.status = "available";

}

Item.prototype.isAvailable = function (){
      return this.status == "available";
}

Item.prototype.issue = function(){
  this.status = "issued";
}

Item.prototype.receive = function(){
  this.status = "available";
}
```

Inheritance

```
function Book(t,a){

  Item.call(this,t);
  this.author = a;

}

Book.prototype = Object.create(Item.prototype);
```

```
var book = new Book("Object-oriented Software
Construction", "Bertrand Meyer");
alert(book.title); // Object-oriented Software Construction
alert(book.author); // Bertrand Meyer
alert(book.isAvailable()); // true
book.issue();
alert(book.isAvailable()); // false
book.receive();
alert(book.isAvailable()); // true
```

class based syntax

```
class Item{
    constructor(t){
        this.title = t;
        this.status = "available";
    }

    isAvailable(){
        return this.status == "available";
    }

    issue(){
        this.status = "issued";
    }

    receive(){
        this.status = "available";
    }
}
```

class based syntax

```
class Book extends Item {

    constructor(t,a){
        super(t);
        this.author = a;
    }
}
```

```
var book = new Book("Object-oriented Software
Construction", "Bertrand Meyer");
alert(book.title); // Object-oriented Software Construction
alert(book.author); // Bertrand Meyer
alert(book.isAvailable()); // true
book.issue();
alert(book.isAvailable()); // false
book.receive();
alert(book.isAvailable()); // true
```

Arrays

- A special kind of object
- Contains a special **length** attribute
  - length = max(numeric_index) + 1;
- Arrays in JavaScript are not contiguous memory locations as in other languages
- Since array is an object, it may contain strings as indexes
  - Similar to:
    - Hashtables
    - Associative arrays in PHP

Arrays

Example 1

```
var a = new Array();
a[0] = 0;
a[5] = 5;
alert(a.length); // ?
```

Example 2

```
var a = [0, 5]; // another way of declaring array
alert(a.length); // ?
```

Built-in Types

- Number
  - No integers, floating points, etc. - Every numeric value is a number
  - 1 = 1.0 = 1e+0
  - Number functions:
    - toExponential
    - toFixed
    - toPrecision
    - toValue
    - toString

Built-in Types

- String
  - Represents a piece of text
- Boolean
  - Represents boolean values (true, false)

- These built-in types like others are also objects and contains numerous functions – reference available on (http://www.w3schools.com)