Darryl Ng
issnes@nus.edu.sg
Institute of System Science
National University of Singapore

Basics

Console object

- global variable console

console.log(msg)	output string to the console window or debug window from browser
console.warn(msg)	prints on stderr
console.time(label)	marks a time stamp
console.timeEnd(label)	prints out the elapsed time since the time function was called
console.assert(cond,message)	throws an AssertionFailure exception if cond evaluates to false

Data type

- undefined
- null
- number
- string
- boolean
- function

```
var y;
console.log(y);

y = null ;
console.log(y);


console.log(typeof 10);
console.log(typeof "hello");
console.log(typeof function () { var x = 20; });
```

Constant and Variable declarations

```
var SECOND = 1 * 1000;

var foo = 'bar';

var keys = ['foo', 'bar'];
var values = [23, 42];

var object = {};
```

Array and Object declarations

```
var a = ['hello', 'world'];

var b = {
  good: 'code',
  'is generally': 'pretty',
};

var c = {};

var user = {
     first_name: "Gloria ",
     last_name: "Ng",
     age: 32,
     website: "www.gloria.com"
};
```

Array Functions

- push()
- pop()
- unshift()
- shift()
- join()
- sort()

```
var nums = [ 1, 1, 2, 3, 5, 8 ];

nums.push(13);
console.log(nums);

nums.pop();
console.log(nums);

nums.unshift(1);
console.log(nums);

nums.shift();
console.log(nums);

var s = nums.join(", ");
console.log(s);

nums.sort();
console.log(nums);
```

Functions – 1
Fundamentals

- fully typed objects
  - manipulated
  - extended
  - passed as data
- function structure

```
function functionName() {
   // function body
   // optional return;
}
```

```
function say(word) {
  console.log(word);
}

function execute(someFunction, value) {
  someFunction(value);
}

execute(say, "Hello");
```

Functions – 2
Immediate Executing Function

- Invoke function wrap in parenthesis ()

```
(function myData() {
   console.log('myData was executed!');
})();
```

Functions – 3
Anonymous Function

- function without a name
- function can be assigned to a variable
- ways of defining inline function

```
var foo1 = function namedFunction() {
   console.log('foo1');
}
foo1(); // foo1

var foo2 = function () { // no function name
i.e. anonymous function
   console.log('foo2');
}
foo2(); // foo2
```

Functions – 4
Higher Order Function

- pass functions to other functions
- functions that take functions as arguments
- **setTimeout** function.

```
function foo() {
    console.log('2000 milliseconds have passed since this demo started');
}
setTimeout(foo, 2000);
```

Functions – 5
Closures

- function defined inside another function
- inner function has access to the variables declared in the outer function
- variables in the outer function have been closed by the inner function
- variables are still bound in the inner function and not dependent on the outer function

```
function outerFunction(arg) {
    var variableInOuterFunction = arg;
    return function () {
       console.log(variableInOuterFunction);
    }
}
var innerFunction = outerFunction('hello closure!');
innerFunction();
```

Error Handling

- use the throw JavaScript keyword
- catch exception with a try / catch block

```
function a () {
    throw new Error("Something bad happened!");
}
try {
    a();
} catch (e) {
    console.log("I caught an error: " + e.message);
}
console.log("program is still running");
```

Classes - 1

- declared as functions
- function that declares the class is its constructor
- all objects in JavaScript have a prototype object (default)
  - mechanism to inherit properties and methods
  - create Inheritance
- use the operator **`instanceof`** to check the inheritance

Classes – 2
Shape Class

```
// Shape - superclass
function Shape () {
}

Shape.prototype.X = 0;
Shape.prototype.Y = 0;

// superclass method
Shape.prototype.move = function (x, y) {
     this.X = x;
     this.Y = y;
}

// superclass method
Shape.prototype.distance_from_origin = function () {
     return Math.sqrt(this.X*this.X + this.Y*this.Y);
}

// superclass method
Shape.prototype.area = function () {
     throw new Error("I don't have a form yet");
}
```

Classes – 3
Square Class

```
// Square - subclass
function Square() {
}

Square.prototype = new Shape();
Square.prototype.__proto__ = Shape.prototype;
Square.prototype.Width = 0;

// override method
Square.prototype.area = function () {
   return this.X * this.Y;
}
```

Classes – 4
Rectangle Class

```
// Rectangle - subclass
function Rectangle() {
  Shape.call(this); // call super constructor.
}

// subclass extends superclass
Rectangle.prototype = Object.create(Shape.prototype);
Rectangle.prototype.constructor = Rectangle;

// Override method
Rectangle.prototype.move = function(x, y) {
  Shape.prototype.move.call(this, x, y); // call superclass method
  log += 'Rectangle moved.\n';
}

// override method
Rectangle.prototype.area = function () {
   return this.X * this.Y;
}
```

Classes – 5
Pattern Usage 1

```
var s = new Shape();
s.move(10, 10);
console.log(s.distance_from_origin());

var sq = new Square();
sq.move(-5, -5);
sq.X = 5;
sq.Y = sq.X;
console.log(sq.distance_from_origin());
console.log(sq.area());

var log = "";
var rect = new Rectangle();
rect.move(20, 20);
rect.X = 5;
rect.Y = 10;
log += ('Is rect an instance of Rectangle? ' + (rect instanceof Rectangle) + '\n'); // true
log += ('Is rect an instance of Shape? ' + (rect instanceof Shape) + '\n'); // true
console.log(log);
console.log(rect.distance_from_origin());
console.log(rect.area());
```

Classes – 5
Pattern Usage 2

```
. . .

console.log(sq instanceof Square);      // true
console.log(sq instanceof Shape);       // true
console.log(sq instanceof Rectangle);   // false
console.log(rect instanceof Rectangle);   // true
console.log(rect instanceof Shape);       // true
console.log(rect instanceof Square);      // false
console.log(sq instanceof Date);        // false
```

Modules – 1
- export & require

- File Based Module System
- three kinds of modules
  - core modules
  - file modules
  - external node_modules.
- export the current module
  - **`module.exports`** variable
- import a module
  - **`require`** function

```
module.exports = function () {
   console.log('a function is called');
};
```

```
var myData = require('./myData');
myData(); // logs out : "a function is called"
```

Modules – 2
export alias

```
var a = function () {
    console.log('a called');
};

var b = function () {
    console.log('b called');
};

module.exports = {
    a: a,
    b: b
};
```

```
module.exports.a = function () {
    console.log('a called');
};

module.exports.b = function () {
    console.log('b called');
};
```

Global object

- variables or members attached to global are available anywhere in the application

```
function printit(var_name) {
  console.log(global[var_name]);
}

global.HTML = "H";
global.CSS = "C";

printit("CSS");
printit("HTML");
printit("SQL");
```

Prompt – 1
Console input

```
var prompt = require('prompt');

 // Start prompt operation
 prompt.start();

 // Get two properties from the user: username and password
 prompt.get(['username', 'password'], function (err, result) {

   // Log the results to commandline console
   console.log('Command-line input received:');
   console.log('  username: ' + result.username);
   console.log('  email: ' + result.password);
 });
```

Prompt – 2
Property settings

- Properties that may be used for validation and prompting controls

```
{
    description: 'Enter your password',  // Prompt displayed to the user.
    type: 'string',                      // Specify the type of input to expect.
    pattern: /^\w+$/,                    // Regular expression to validate input field.
    message: 'Password must be letters', // Warning message to display if validation fails.
    hidden: true,                        // characters entered will not be output to console
    replace: '*',                        // Replace each hidden character with specified string.
    default: 'lamepassword',             // Default value to use if no value is entered.
    required: true                       // If true, value entered must be non-empty.
    before: function(value) { return 'v' + value; } // Runs before node-prompt callbacks.
}
```

Control Statements

- for
- foreach
- while
- do..while
- if..else
- switch..case
- break

Loop statement – 1
for / forEach

```
var my_array = ['a', 'b', 'c'];

for (var i=0; i<my_array.length; i++) {
        console.log(my_array[i]);
        //a b c
}

my_array.forEach(function(current_value) {
        console.log(current_value);
        //a b c
});
```

Loop statement – 2
while / do..while

```
var products = [
  { name: 'Running shoes', price: 75 },
  { name: 'Golf shoes',    price: 85 },
  { name: 'Dress shoes',   price: 95 },
  { name: 'Walking shoes', price: 65 },
  { name: 'Sandals',       price: 55 }
];

var i = 0;

while (i < products.length) {
        console.log(products[i].name);
        i++;
}
```

Loop statement – 3 if..else

```
var products = [
  { name: 'Running shoes', price: 75 },
  { name: 'Golf shoes',    price: 85 },
  { name: 'Dress shoes',   price: 95 },
  { name: 'Walking shoes', price: 65 },
  { name: 'Sandals',       price: 55 }
];

var i = 0;

while (i < products.length) {
       if (products[i].price > 80) {
              console.log(products[i]);

       }
       i++;
}
```

Operators

```
var a = 0;
if (a === '') {
  console.log('winning');
}
```

```
var isCat = true;
var pet = isCat ? "cat" : "dog";
console.log(pet);
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Comparison_Operators

Loop statement – 4 switch..case

```
var prompt = require('prompt');

var products = [
  { name: 'Running shoes', price: 75 },
  { name: 'Golf shoes',    price: 85 },
  { name: 'Dress shoes',   price: 95 },
  { name: 'Walking shoes', price: 65 },
  { name: 'Sandals',       price: 55 }
];
```

```
prompt.get([{
    name: 'Product',
    description: 'Enter value 0..4',
    type: 'string',
    required: true
  }], function(err, results) {
    switch(results.Product) {
      case '0':
             console.log(products[0]); break;
      case '1':
             console.log(products[1]); break;
      case '2':
             console.log(products[2]); break;
      case '3':
             console.log(products[3]); break;
      case '4':
             console.log(products[4]); break;
      default:
      console.log('Wrong input defined!');
             break;
    }
});
```

Callback Function - 1

- Synonymous to asynchronous
- function to read a file may start reading file and return the control to the execution environment immediately so that the next instruction can be executed
- Callback function called on completion
- Result returns as parameter
- No blocking I/O
- highly scalable
- Process high number of requests without waiting for any function to return results

Callback Function - 2

```
var fs = require("fs");

var data = fs.readFileSync('example.txt');

console.log(data.toString());
console.log("Program Ended");



var fs = require("fs");

fs.readFile(example.txt', function (err, data) {
   if (err) return console.error(err);
   console.log(data.toString());
});

console.log("Program Ended");
```

File System – 1

```
// Load the fs (filesystem) module
var fs = require('fs');

// Read the contents of the file into memory.
fs.readFile('example.txt', function (err, logData) {

  // If an error occurred, throwing it will
  // display the exception and end our app.
  if (err) throw err;

  // logData is a Buffer, convert to string.
  var text = logData.toString();

  console.log(text);
});
```

File System - 2

```
// Read the contents of the file into memory.
fs.readFile('example.txt', function (err, logData) {

  // If an error occurred, throwing it will display the exception and end our app.
  if (err) throw err;

  // logData is a Buffer, convert to string.
  var text = logData.toString();
  var results = {};

  // Break up the file into lines.
  var lines = text.split('\n');
  lines.forEach(function(line) {
    var parts = line.split(' ');
    var letter = parts[1];
    var count = parseInt(parts[2]);

    if(!results[letter]) {
      results[letter] = 0;
    }
        results[letter] += parseInt(count);
  });

  console.log(results);
});
```

OS Module

- operating-system related utility functions and properties

```
var os = require('os');
var gigaByte = 1 / (Math.pow(1024, 3));
console.log('Total Memory', os.totalmem() * gigaByte, 'GBs');
console.log('Available Memory', os.freemem() * gigaByte, 'GBs');
console.log('Percent consumed', 100 * (1 - os.freemem() / os.totalmem()));
```

Util Module

- log function log to console with timestamp
- format function similar to C printf function
  - Placeholders: %s (strings) and %d (numbers)
- check particular type (isArray, isDate, isError)

```
var util = require('util');
util.log('sample message');

var name = 'CSS';
var a = 33;

console.log(util.format('%s has %d attributes', name, a));

console.log(util.isArray([])); // true
console.log(util.isArray({ length: 0 })); // false

console.log(util.isDate(new Date())); // true
console.log(util.isDate({})); // false

console.log(util.isError(new Error('This is an error'))); // true
console.log(util.isError({ message: 'I have a message' })); // false
```

Buffering

- manipulate Binary Data with Buffers
- streams and files
- hold binary data that can be converted into other formats

```
var b = new Buffer(10000);
var str = "We want to go visit and tour around the world in 80 days.";
b.write(str); // default is utf8, which is what we want
console.log( b.length ); // will print 10000 still!

console.log( str.length );                 // prints XX characters size
console.log( Buffer.byteLength(str) );     // prints XX characters size
```

Delay Function

- sets up a function to be called after a specified delay in milliseconds
- `setTimeout()` function
- `setInterval()` function

```
var count = 0;

setTimeout(function () {
    count++;
    console.log(‘hello world! ‘ + count);
}, 1000);
```

```
var count = 0;

var intervalObject = setInterval(function () {
    count++;
    console.log(‘hello world! ‘ + count);

    if (count == 5) {
        console.log('exiting');
        clearInterval(intervalObject);
    }
}, 1000);
```