Javascript: the Basics

Shan-Hung Wu and DataLab
CS, NTHU

HTML vs. CSS vs. Javascript

- HTML defines content and element structure
  - The “Nouns”
- CSS defines how an element looks
  - The “Adjectives”
- Javascript defines how an element interact with users
  - The “Verbs”

Javascript

- An implementation of ***ECMAScript*** (ES) standard
  - Javascript 1.5, 1.7, 1.8.5 are non-official standards maintained by Mozilla
- ES5 = ECMAScript 5 (2009)
  - Supported by major browsers
  - Covered by this class
- ES6 = ECMAScript 6 = ES2015
- ES7 = ECMAScript 7 = ES2016
  - Not fully supported yet
  - Luckily, transpilers such as Babel are avalable
  - To be covered in the next class

Running Javascript in Browsers

- In *.js files

```
window.onload = function() {
  var el = document.querySelector('h1');
  el.textContent = 'Hello Javascript!';
};
```

  - When loading HTML, code inside `<script>` is executed ***immediately***

- In Chrome console

```
console.log(el.textContent);
```

Observations

```
window.onload = function() {
  var el = document.querySelector('h1');
  el.textContent = 'Hello Javascript!';
};
```

- Statements and expressions similar to C
- No `main()`, but there is a global scope
- There are built-in objects (`window`, `document`)
  - An object is like `struct` in C
- Variables (`el`) have no type
- Functions are first-class citizens
- Interacts with user/browser via events and handlers

Outline

- Variables and Types
- Expressions and Control Flows
- Built-in Functions and Methods
- DOM and Event Handling
- Tricky Parts: `this` and Closures

Outline

- Variables and Types
- Expressions and Control Flows
- Built-in Functions and Methods
- DOM and Event Handling
- Tricky Parts: `this` and Closures

Variables

```
var i = 7;
var pi = 3.1416;
var name = 'Rusty';
var isFun = true;
```

- Store values provided by ***literals***
- Not tied to specific type
- Use `typeof` to determine the type

```
i = 'abc';
typeof i // 'string'
```

Types

- 5 primitive types:
  - Number, string, boolean
  - Undefined, null
- 2 object types:
  - Object, function

Numbers

```
/* literals */
4
9.3
-10
NaN

/* expressions */
4 + 10
1 / 5      // 0.2
10 % 3     // 1
-10 % 3    // -1
```

Strings

```
/* literals */
'hello world'
"hello world"
"My name is \"Bob\""
'My name is "Bob"'
'This is backslash: \\'

/* expressions */
'wi' + 'fi'  // 'wifi' (new string)
'hello'[0]   // 'h'
'hello'[4]   // 'o'

/* properties */
'ti ta'.length      // 5
'hello'.slice(3,5)  // 'lo'
```

- Immutable

Booleans

```
/* expressions */
true && false  // false
true || true   // true
!true          // false
```

- ***Short-circuit*** evaluation

```
false && true
true || false
```

Undefined vs. Null

```
/* implicit empty */
var i;
typeof i // 'undefined'


/* explicit empty */
var i = null;
typeof i // 'object' ('null' in ECMAScript)
```

Objects I

```
var name = 'John';

/* literal (JSON) */
var user = {
  name: 'Bob',
  friends: ['Alice', 'Paul'], // array
  greet: function() { // method
    return 'Hi, I am ' + this.name;
  }
};

user.name     // 'Bob' (not 'John')
user['name'] // 'Bob'
user.greet() // 'Hi, I am Bob'
```

- Like `struct` in C
- But have ***methods***

Objects II

```
/* arrays */
var arr = [7, 'b', [false]];
var arr = new Array(7, 'b', [false]);
arr[1]        // 'b'
arr.length    // 3


/* dates */
var now = new Date();
now.toUTCString()
var d = new Date('March 1, 1997 11:13:00');

/* regexps */
var re = /ab+/i;
var re = new RegExp('ab+', 'i');
re.test('Abbbc')                    // true
re.test('bcd')                      // false
'Abbc abc'.replace(/ab+/ig, 'x') // 'xc xc'
```

- Arrays, dates, regexps are special kinds of objects

Functions

```
/* functions */
function add(a, b) {
  return a + b;
}
var add = function(a, b) {
  return a + b;
}; // anonymous function
add(1, 3)     // 4
add('Hi!')    // Hi!undefined

function add() {
  return arguments[0] + arguments[1];
}

/* high-order functions */
function forEach(arr, f) {
  for(var i = 0; i < arr.length; i++) f(arr[i]);
}
forEach(['a', 'b', 'c'], console.log); // no ()
```

- Functions are callable objects
- First-class citizens

Functions as Methods

```
function greet() {
  return 'Hi, I am ' + this.name;
}
greet()        // 'Hi, I am undefined'
```

- `this` is the context of execution
  - `window` by default

```
var user = {
  name: 'Bob'
};
user.greet = greet;
user.greet()  // 'Hi, I am Bob'
```

Functions as Constructors

```
function User(name, friends) {
  this.name = name;
  this.friends = friends;
  this.greet = function() {...};
};
// saves repeated code
var user1 = new User('Bob', [...]);
var user2 = new User('John', [...]);

typeof User     // 'function'
typeof user2    // 'object'
```

- `new` creates an empty object, calls constructor, and then returns the object
- `User` is called a ***class***
  - Blueprint for its objects/***instances***

Identifying Classes

- How to tell the class of an object?

```
[1,2,3,4].constructor              // Array function
{name:'Bob',age:34}.constructor  // Object function
new Date().constructor             // Date function
function(){}.constructor           // Function function


function isDate(obj) {
    return obj.constructor == Date;
}
```

Static Methods

- Methods (of a class) that do not require an instance to run
  - No `this` inside
- Convention: defined in the constructor
  - Recall that a constructor (function) is an object

```
Math.round(4.7)      // 5
Math.floor(4.7)      // 4
Math.pow(8, 2)       // 64
Math.sqrt(64)        // 8
Math.sin(Math.PI)    // 1
```

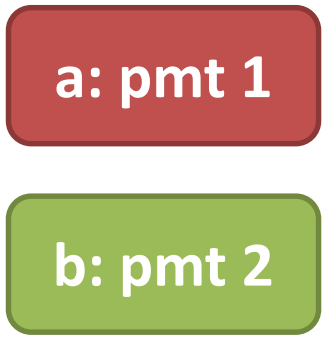
  - `Math` does not allow instances so it is just an object

Primitives vs. Objects I

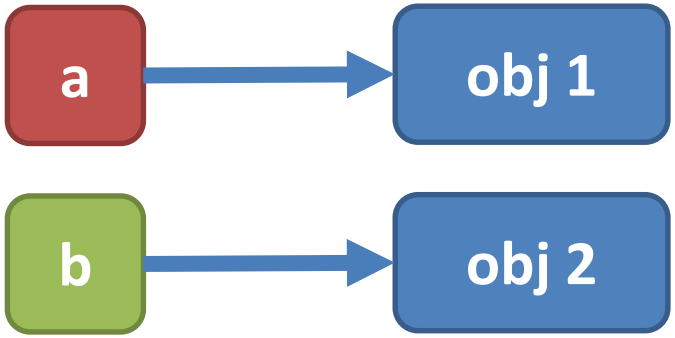
- Both primitives and objects can have properties and methods
  - But no custom members for primitives

```
var str1 = 'Hello!';                // primitive
var str2 = new String('Hello!'); // object
str1.slice(3, 5)  // 'lo'
str2.slice(3, 5)  // 'lo'
var f = function() { ... };
str1.method = f;
str2.method = f;
typeof str1.method // 'undefined'
typeof str2.method // 'function'
```

Primitives vs. Objects II




```
var a = ...;
var b = ...;
```




```
var a = b;
```

```
function f(b) {b++;}
var a = ...;
f(a)
```

```
if (a == b) {...}
```

Type	Assigned by	Passed by	Compared by
boolean	Value	Value	Value
number	Value	Value	Value
string	Immutable	Immutable	**Value**
object/function	Reference	Reference	Reference

String Comparison

```
var s = 'abc';
var s2 = s.slice(); // 'abc' (a clone)
s == s2             // true

var s3 = new String(s);  // object
var s4 = new String(s);  // object
s3 == s4                 // false
```

Naming Convention

- Variables: lower camel case, start with letter
  - E.g., `variableName`
- Constants: upper case with separator '_'
  - E.g., `CONSTANT_NAME`
- Functions: lower camel case
  - E.g., `functionName`
- Classes/constructors: upper camel case
  - E.g., `ClassName`

Outline

- Variables and Types
- Expressions and Control Flows
- Built-in Functions and Methods
- DOM and Event Handling
- Tricky Parts: `this` and Closures

Expression Evaluation

```
3 + 12 / 2 / 3          // 5
```

- ***Precedence***: order in which operators are evaluated
- ***Associativity***: order in which operators of the same precedence are evaluated

- See Precedence & Associativity Table

Control Flows I

```
if (exp) {
   ...
} else if (exp) {
   ...
} else {
   ...
}

while (exp) {
   ...
}
do {
   ...
} while (exp);
```

- Similar to those in C

```
for (var i = 0; i < 5; i++) {
  ...
}
for (var prop in obj) {
  obj[prop]...
}

switch (num or string) {
  case cat':
    ...
    break;
  case dog':
    ...
    break;
  default:
    ...
}
```

Control Flows II

Truthy and Falsy Values

```
if (exp) {...}
```

- `exp` should be a Boolean expression
- However, non-Boolean values can be implicitly "truthy" or "falsy"
- Try these expressions:

```
!!'Hello world!'
!!''
!!null
!!0
!!-5
!!NaN
```

Falsy Values

```
false
0
''
null
undefined
NaN
```

- Everything else is truthy

Equality Operators

```
'' == '0'           // false
'' == 0             // true
'0' == 0            // true
' \t\r\n' == 0      // true
false == 'false'    // false
false == 0          // true
false == undefined  // false
false == null       // false
null == undefined   // true
NaN == NaN          // false
```

- Use === (!==) instead of == (!=)
- == does not check the type of operands
- All above expressions return false with ===

Variable Scopes

- Global/window scope
- Function scope

```
var i = 7;

function f() {
  var j = 10;
  i          // 7
  window.i // 7
}

j  // undefined
```

- ***No block scope*** in ES5

```
function f() {
  for(var i = 0; i < 10; i++) {
    ...
  }
  ...
  var i; // i equals 10
}
```

Outline

- Variables and Types
- Expressions and Control Flows
- Built-in Functions and Methods
- DOM and Event Handling
- Tricky Parts: `this` and Closures

Type Conversion

```
/* to strings */
String(123)          // '123'
(123).toString()     // '123'
(12.345).toFixed(2) // '12.35'
String(false)        // 'false'
false.toString()     // 'false'

/* to numbers */
Number('3.14')       // 3.14
Number(true)         // 1
Number('')           // 0
Number('99 1')       // NaN
```

Alerts and Prompts

```
var name = prompt('Enter your name:');
console.log('Entered: ' + name);
alert('Hello ' + name + '!');
```

- Exercise: Guess Game
  - ‘Guess a number’
  - ‘To large/small, guess again’
  - ‘Correct!’

Timers

```
function tick() {
  console.log(new Date().getSeconds());
}

/* call tick every 1000ms */
var id = setInterval(tick, 1000);

/* stop calling */
clearInterval(id);
```

JSON

```
var user = {
  user: 'Bob',
  friends: ['Alice', 'John']
};
var json = JSON.stringify(user); // string

var user2 = JSON.parse(json);
user === user2                   // false
```

Arrays I

```
var arr = ['r', 'g', 'b'];

/* stack */
var b = arr.pop();     // ['r', 'g']
arr.push('y');         // ['r', 'g', 'y']

/* queue */
var r = arr.shift();   // ['g', 'y']
arr.unshift('y');      // ['y', 'g', 'y']

/* loop */
function f() {...}
arr.forEach(f);   // high-order function
```

Arrays I

```
var arr = ['r', 'g', 'b', 'g'];

arr.indexOf('g')     // 1, not 3
arr.indexOf('m')     // -1

/* copy */
var arr2 = arr.slice(1, 3);  // ['g', 'b']
arr.length                   // 4

/* remove */
var arr3 = arr.splice(1, 2); // ['g', 'b']
arr.length                   // 2
```

Strings

- Has `length`, `indexOf()`, `slice()`
- No `splice()` since immutable

```
var str = 'Please locate where "locate" is.';

str.charAt(0)              // 'P'
str[0]                     // don't do this
str.search('locate')       // 7

var str2 = str.replace(/locate/g, 'find')
str2   // 'Please find where "find" is.'
str1   // 'Please locate where "locate" is.'

var srt3 = str.toUpperCase()
str.split(' ').length    // 5
```

Outline

- Variables and Types
- Expressions and Control Flows
- Built-in Functions and Methods
- DOM and Event Handling
- Tricky Parts: `this` and Closures

http://www.patatap.com

Built-in Objects

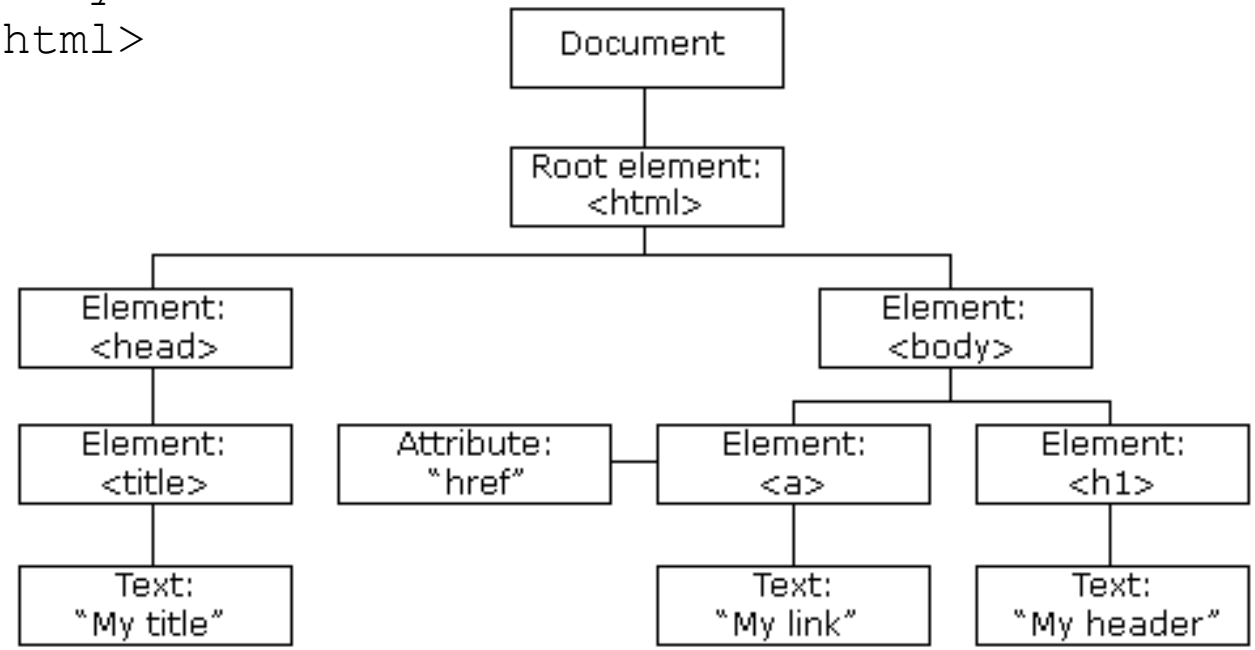
- Browser objects:
  - `navigator`: info. about the browser
  - `window`: an opened window/tab
  - `location`: info. about current URL
- ***DOM*** (Document Object Model) objects:
  - `document`: the HTML document

- Use `console.dir(obj)` to see members

DOM

```
<html>
<head>
  <title>My title</title>
</head>
<body>
  <h1>My header</h1>
  <a href="...">My link</a>
</body>
</html>
```

- Interface between JS and HTML/CSS
  - Every HTML element is an `Element` object
  - A tree
  - Text and attribute nodes



DOM Manipulation Process

- ***Select*** and then ***manipulate***

```
<html>
<head>
  <title>My title</title>
</head>
<body>
  <h1>My header</h1>
  <a href="...">My link</a>
</body>
</html>
```

```
var el = document.querySelector('h1');
el.style.color = 'red';
```

DOM Selectors

```
document.URL
document.documentElement()    // <html>
document.head                 // <head>
document.body                 // <body>
document.links
```

```
document.querySelector()      // returns first match
document.querySelectorAll()   // returns a list
document.getElementById()
document.getElementsByCalssName()
document.getElementsByTagName()
```

DOM Manipulation

```
var el = document.querySelector(...);

el.style.backgroundColor = 'blue';
el.style.border = '0.25rem solid red';

el.classList.add('some-class');
el.classList.remove('some-class');
el.classList.toggle('some-class');

el.textContent = 'Some text';    // no HTML tag
el.innerHTML = 'Some HTML fragment';

el.getAttribute('href')
el.setAttribute('src', 'http://...')
```

Form Manipulation

```
<html>
<body>
  <form id="user-form">
    <input type="text" name="email" />
    <select name="sex">
      <option value="male" selected="selected">Male</option>
      <option value="female">Female</option>
    </select>
    <select name="major" multiple='multiple' >
      <option value="math">Math</option>
      <option value="cs">CS</option>
      <option value="ee">EE</option>
    </select>
    <input type="radio" name="grade" value="A" />
    <input type="radio" name="grade" value="B" />
    <input type="checkbox" name="valid" value="valid" />
    ...
  </form>
</body>
</html>
```

```
var formEl = document.getElementById('user-form');
var emailEl = formEl.elements['email'];
alert(mailEl.value);
var sexEl = formEl.elements['sex'];
alert(sexEl.options[sexEl.selectedIndex].value);
var majorEl = formEl.elements['major'];
for(var i = 0; i < majorEl.options.length; i++) {
  if(majorEl.options[i].selected)
    alert(majorEl.options[i].value);
}
var gradeEls = formEl.elements['grade'];
for(var i = 0; i < gradeEls. length; i++) {
  if(gradeEls[i].checked)
    alert(gradeEls[i].value);
}
var validEl = formEl.elements['valid'];
if(validEl.checked) alert(validEl.value);
```

Event Handling

```
el.onclick = function(e) {
  ...
};              // replace previous handler

el.addEventListener('click', function(e) {
  ...
});             // can have multiple handlers
```

Event Types

```
el.addEventListener('click', function(e) {
  ...
});
```

- 300+ types available, e.g., `'contextmenu'`, `'mouseover'`, `'mouseout'`, `'dbclick'`, `'keypress'`, `'drag'`, `'submit'`, etc.
- Exercise:

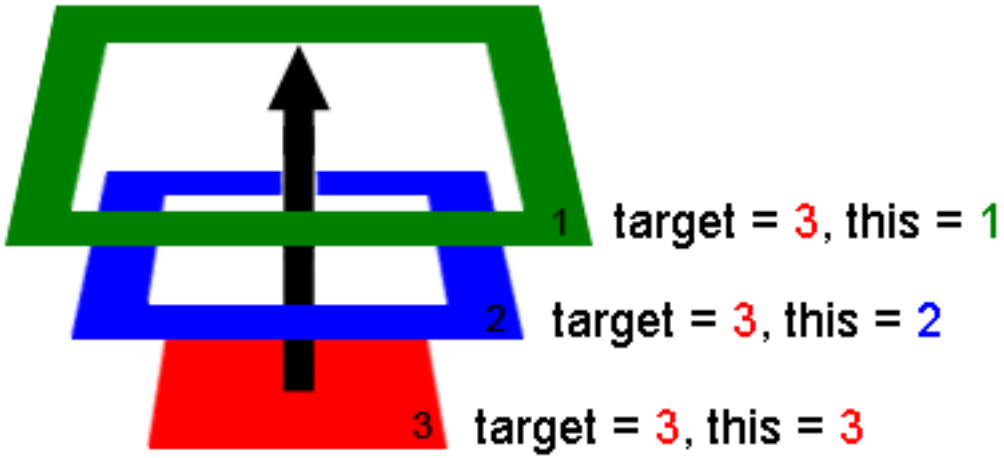
Event Objects

```
el.addEventListener(..., function(e) {
   ...
};
```

Props/Methods	Description
clientX/Y	Mouse coordinates (relative to upper-left corner of the window) at event time
type	Type indicator as string, e.g., “mouseover”, “click”, etc.
currentTarget	Element to which the current handler is assigned
target	Element that triggers the event. Not necessary the one to which the handler is assigned
relatedTarget	Secondary element. On “mouseover” (resp. “mouseout”), indicates the element that the mouse has left from (resp. moved into)
preventDefault()	Cancels any default action associated with the event
stopPropagation()	Prevent the event from bubbling

Event Bubbling

```
<div id="1">
  <div id="2">
    <div id="3">
      ...
    </div>
  </div>
</div>
```



- An event “bubbles up” to the root
- Use `e.target` to access the originator
- Use `this` or `e.currentTarget` to access the element to which the handler attaches
- To stop: `e.stopPropagaton()`

Canceling Default Handlers

- Sometimes, you may want to cancel the default browser behavior for an event
  - E.g., to prevent `<form>` from sending HTTP request if validation fails
- To cancel: `e.preventDefault()`
  - Does ***not*** stop event bubbling

Outline

- Variables and Types
- Expressions and Control Flows
- Built-in Functions and Methods
- DOM and Event Handling
- Tricky Parts: `this` and Closures

Be Careful about `this`

```
var user = {
  name: 'Bob',
  greet: function() {
    console.log('Hi, I am ' + this.name);
  }
};
setInterval(user.greet, 1000);
```

- Output?
- `this` always binds to the "current owner" of that function
  - **`window`**, when called as a function
  - ***Object to which . is applied***, when called as a method
  - ***Creating object***, when called as a constructor

Explicit Binding

```
var user = {
  name: 'Bob',
  greet: function() {
    console.log('Hi, I am ' + this.name);
  }
};
setInterval(user.greet.bind(user), 1000);

// 'Hi, I am Bob'
```

Call with Explicit Binding

```
var user = {
  name: 'Bob',
  greet: function(peer) {
    console.log('Hi ' + peer + ', I am ' + this.name);
  }
};
var user2 = {
  name: 'Alice'
};

// 'Hi John, I am Alice'
user.greet.call(user2, 'John');
user.greet.apply(user2, ['John']); // for delegation

function greetFromAlice() {
  return user.greet.apply(user2, arguments);
}
greetFromAlice('Paul') // 'Hi Paul, I am Alice'
```

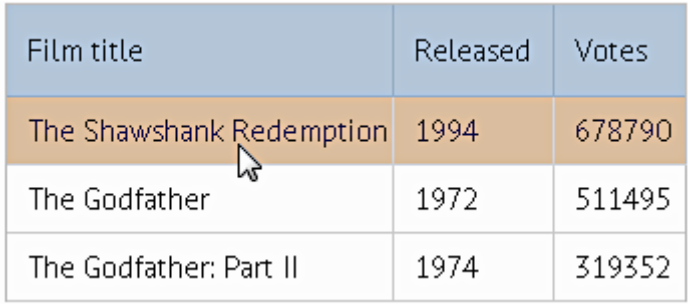
Closures

```
/* high-order function */
function createShift(i) {
  return function(j) { // closure
    return i + j;
  }
}
var shift = createShift(100);
shift(3) // 103
```

- ***Closure*** is a function using variables defined in outer function that has returned
- If it accesses data outside, those data are kept in memory (after outer function returns)

Bad Closures

```
var trs = document.querySelectorAll('tr');
for(var i = 0; i < trs.length; i++) {
  var tr = trs[i];
  tr.onmouseover = function() {
    tr.classList.add('row-over');
  };
}
```



Film title	Released	Votes
The Shawshank Redemption	1994	678790
The Godfather	1972	511495
The Godfather: Part II	1974	319352

- All handlers add class to the same last `<tr>`
- Fix?

```
var trs = document.querySelectorAll('tr');
for(var i = 0; i < trs.length; i++) {
  var tr = trs[i];
  tr.onmouseover = function() {
    this.classList.add('row-over');
  };
}
```

Good Closures

```
var trs = document.querySelectorAll('tr');
for(var i = 0; i < trs.length; i++) {
  var data = ... // based on trs[i]
  tr[i].onmouseover = function() {
    ... // process data
  };
}
```

- Fix?

```
var trs = document.querySelectorAll('tr');
for(var i = 0; i < trs.length; i++) {
  var data = ... // based on trs[i]
  tr[i].onmouseover = (function(d) {
    return function() {
      ... // process d
    }
  })(data);
}
```

- Use closures to create “private” variables

Assigned Readings

- Re-introducing Javascript
- Regular expression in Javascript (optional)
- More about closures (optional)

Demo: Color Game