# JavaScript - Basic Programming

## CPEN322

*The Univerity of British Columbia*
Department of Electrical and Computer Engineering
Vancouver, Canada


Electrical and Computer Engineering


January 31, 2024

# Including Javascript

# Including Javascript (1)



- 1) Directly in the HTML page

```
1  <html>
2      <head>
3          <title>My JavaScript Page</title>
4      </head>
5      <body>
6          <script type="text/javascript">
7              var i = 2+2;
8              document.writeln(i);
9          </script>
10     </body>
11 </html>
```

# Including Javascript (2)



- 2) In an external ".js" file

```
1  <html>
2      <head>
3          <title>My JavaScript Page</title>
4          <script type="text/javascript" src="myscript.js"></
               script>
5      </head>
6      <body>
7          ...
8      </body>
9  </html>
```

# Basic Constructs

# Comments



- Useful to document your Javascript code!
  - Any line starting with `//` is ignored
  - The *right part* of any line containing with `//` is ignored
  - Everything between `/* and */` is ignored (useful for multi-line comments)

```
1  // This line will be ignored by the Javascript engine
2
3  var x = 2; // This part of the line will be ignored
4
5  /* These lines will
6  be ignored by the
7  Javascript engine */
```

# Variables - Declaration



- Use the var keyword to declare **local** variables, which hold data in your program
- "Duck typing": no need to specify type of variables (as in Java, C++, C#, etc.) $\Rightarrow$ similar to Python
- Any variable can be assigned any value

```
1  var foo = 0;
2  // foo = 0
3
4  foo = foo + 2;
5  // foo = 2
6
7  foo = "My name is ";
8  foo += "Julien";
9  // foo = "My name is julien"
10
11 var bar = foo + ":-)"
12 // bar = "My name is julien :-)"
```

# Variables - Arithmetic Operators



- Assignment: $=$ $\Rightarrow$ set the value of a variable
- Basic arithmetics: $+$, $-$, $*$, $/$, % (modulo), () (ordering)
- Incrementation: $+=$, $-=$, $*=$, $/=$
  - foo $+= 1$ $\Rightarrow$ foo $=$ foo $+ 1$
- Pre/post incrementation: foo++, ++foo

```
1  var foo = 5;
2  foo = foo + 1 - 2 * (4 - 1);
3  // foo = ???
4
5  var bar = 4;
6  bar += bar++;
7  // bar = 8
8
9  var baz = 4;
10 baz += ++baz;
11 // bar = ???
```

# Functions



- Wrapping common behavior
- Avoiding code repetition
- Providing *abstractions* : no need to understand the internals of the function if the definition is clear

**Function Definition**

- **Name**
- **Inputs**
- **Output**
- Body

```
1  function areaOfCircle(radius) {
2      var PI = 3.1416;
3      return PI * square(radius);
4  }
5
6  function square(x) {
7      return x*x;
8  }
9
10 var A = areaOfCircle(2);
```

# Functions - Nesting



- In Javascript, functions can be nested (unlike in C or Java)
- A nested function is a function defined in another
- Example below: square can only be invoked from within areaOfCircle

```
1  function areaOfCircle(radius) {
2      var PI = 3.1416;
3
4      function square(x) {
5          return x*x;
6      }
7
8      return PI * square(x);
9  }
10
11 var A = areaOfCircle(2);
```

# Scope of Variables



- Global scope: variable usable by all JS code executed in the web page context (A)
- Local scope: variable usable within a function and sub-functions (PI, sq)
- Parameters: usable only within their own function (radius, x)

```
1  function areaOfCircle(radius) {
2     var PI = 3.1416;
3
4     // This is a Nested function
5     function PIsquare(x) {
6        var sq = x * x;
7        return PI * sq;
8     }
9
10    return PIsquare(radius);
11 }
12
13 var A = areaOfCircle (2);
14 console.log("Area of circle of radius 2 = " + A);
```

# var, let, const



- **var**:
  - scope: function
  - re-declaration: allowed within its scope
  - Mutable: Yes
- **let**:
  - scope: block
  - re-declaration: not allowed in same scope
  - Mutable: Yes
- **const**:
  - scope: block
  - re-declaration: Not allowed
  - Mutable: No

# Simple Arrays (1)



- Flexible mechanism allowing to declare/define multiple elements at once
- Problematic code - complete lack of flexibility:

```
1  var vspResult1 = 99;
2  var vspResult2 = 96;
3  var vspResult3 = 93;
4  var vspResult4 = 91;
5  //...
6  var vspResult36 = 41;
```

- Using arrays:

```
1  var vspResults = [99, 96, 93, 91, /* ... */, 41];
2  /* Printing the grade of the top 3 students -- be careful,
3     in most programming languages, the first index is 0! */
4  console.log("Grade of the 1st student: " + vspResults[0]);
5  console.log("Grade of the 2nd student: " + vspResults[1]);
6  console.log("Grade of the 3rd student: " + vspResults[2]);
```

# Simple Arrays (2)



- Adding an item to the end of an array:

```
1  var vspResults = [99, 96, 93, 91, /* ... */, 41];
2  vspResults.push(39);
```

- Removing an item at the end of an array:

```
1  var vspResults = [99, 96, 93, 91, /* ... */, 41, 39];
2  vspResults.pop(); // Removes 39
```

- Getting length of an array:

```
1  var vspResults = [99, 96, 93, 91, /* ... */, 41];
2  console.log( vspResults.length );
```

# Conditionals

# Boolean Expressions and Operators



- Condition that evaluates to true or false
- Operators:
  - Equals: ==
  - Different than: !=
  - Greater than: >
  - Greater than or equal to: >=
  - Smaller than: <
  - Smaller than or equal to: <=
- In addition to:
  - Equals and same type: ===
  - Different than or different type: !==

# Boolean Expressions and Operators - Example



## Exercise

```
1  var x = 5;
2
3  console.log(x == 5);      // ???
4  console.log(x != 4);      // ???
5  console.log(x > 5);       // ???
6  console.log(x >= 5);      // ???
7  console.log(x < 5);       // ???
8  console.log(x <= 5);      // ???
9
10 console.log(x === 5);     // ???
11 console.log(x === "5");   // ???
12 console.log(x !== 5);     // ???
13 console.log(x !== "5");   // ???
14
15 var foo = "VSP";
16 console.log(foo == "VSP");     // ???
17 console.log(foo === "VSP");    // ???
18 console.log(foo != "UBC");     // ???
19 console.log(foo !== "42");     // ???
```

# Boolean Expressions and Operators - Example



## Solution to exercise

```
1  var x = 5;
2
3  console.log(x == 5);      // true
4  console.log(x != 4);      // true
5  console.log(x > 5);       // false
6  console.log(x >= 5);      // true
7  console.log(x < 5);       // false
8  console.log(x <= 5);      // true
9
10 console.log(x === 5);     // true - equals+same type
11 console.log(x === "5");   // false - different type
12 console.log(x !== 5);     // false - not equals
13 console.log(x !== "5");   // true - different type
14
15 var foo = "VSP";
16 console.log(foo == "VSP");     // true
17 console.log(foo === "VSP");    // true
18 console.log(foo != "UBC");     // true
19 console.log(foo !== "42");     // true
```

# Combined Boolean Operators



- x && y: true if both x and y are true
- x || y: true if at least x or y is true
- !x: true if x is false!
- Parentheses are allowed!

## Exercise

```
1  // Returns true if value >= min and value <= max
2  function isBetween(value, min, max) {
3      return ( /* ... */ );
4  }
```

# Combined Boolean Operators



- x && y: true if both x and y are true
- x || y: true if at least x or y is true
- !x: true if x is false!
- Parentheses are allowed!

## Solution to exercise

```
1  // Returns true if value >= min and value <= max
2  function isBetween(value, min, max) {
3      return ( (value >= min) && (value <= max) );
4  }
```

# If Statements (1)



- Execute code if a condition is true and if condition is false (optional)
- condition is any boolean expression

```
1  if (condition) {
2      // Code if condition is true
3  }
4
5  if (condition) {
6      // Code if condition is true
7  } else {
8      // Code if condition is false
9  }
```

- If we omit the { and } symbols, we are only allowed one statement after the if / else!

```
1  if (condition)
2      console.log("This line executes if condition is true");
3  console.log("This line will ALWAYS execute");
```

# If Statements (2)



```
1  // Returns true if value >= min and value <= max
2  function isBetween(value, min, max) {
3      return ( (value >= min) && (value <= max) );
4  }
5
6  if ( isBetween(15, 10, 20) ) {
7      console.log("Number within range!");
8  } else {
9      console.log("Number not within range!");
10 }
```

Is equivalent to:

```
1  if ( (15 >= 10) && (15 <= 20) ) {
2      console.log("Number within range!");
3  } else {
4      console.log("Number not within range!");
5  }
```

# If Statements (3)



- If-statements can be chained

```
1  // Returns true if value >= min and value <= max
2  function isBetween(value, min, max) {
3      return ( (value >= min) && (value <= max) );
4  }
5
6  if ( isBetween(15, 10, 20) ) {
7      if ( isBetween(15, 14, 16) ) {
8          console.log("Excellent!");
9      } else {
10         console.log("Good.");
11     }
12 } else {
13     console.log("Bad!");
14 }
```

# If Statements (4)



- A common programming trick is to chain else conditions

```
1  var score = 75;
2  var grade = "";
3
4  if ( score >= 80 ) {
5      grade = "A";
6  } else if (score >= 70) {
7      grade = "B";
8  } else if (score >= 60) {
9      grade = "C";
10 } else if (score >= 50) {
11     grade = "D";
12 } else {
13     grade = "F";
14 }
15
16 console.log("Your grade is " + grade);
```

# Loops



- Mechanism for repeating (iterating) a portion of code multiple times, until a condition becomes false
- Syntax very similar to Java / C / C#
- Types of loops:
  - For: typically for repeating *n* times
  - While: repeat as long as (while) condition is true. If condition is initially false, no iteration will occur.
  - Do while: repeat as long as (while) condition is true. A first iteration is always guaranteed to occur, even if condition is initially false.
  - For in: for iterating over arrays, collections of objects etc. (to be seen later)

# For Loops (1)



```
1  for (initial_condition; condition; increment) {
2      // Do stuff...
3  }
```

- Steps:
  - Setup initial condition (variable)
  - If condition is true, then execute the inner portion of the loop; otherwise, exit the loop
  - After executing the inner portion of the loop, execute the increment portion (increment loop variable)

# For Loops (2)



- We usually use *i* as a for loop variable. In a nested loop, we can use *j* (and even *k*).
- The initial condition is to usually assign the start value (i.e, 0) to the loop variable
- The increment portion of the loop usually consists of an incrementing operator such as i++ or i+=2
- Example - printing top 3 results:

```
1  var vspResults = [99, 96, 93, 91, /* ... */, 41];
2  var i;
3  for (i = 0; i < 3; i++) {
4      console.log("Score #" + (i+1) + ": " + vspResults[i]);
5  }
```

- We usually declare the loop variable in the initial condition:

```
1  for (var i = 0; i < 3; i++) {
2      console.log("Score #" + (i+1) + ": " + vspResults[i]);
3  }
```

# For Loops (3)



- Complex boolean conditions are supported
- Exercise: printing top results, but stop when they get below 90:

## Exercise

```
1  var vspResults = [99, 96, 93, 91, /* ... */, 41];
2
3  // Change the following loop to print the
4  // results but stop when they get below 90:
5  for (var i = 0; i < 3; i++) {
6      console.log("Score #" + (i+1) + ": " +
           vspResults[i]);
7  }
```

# For Loops (3)



- Complex boolean conditions are supported
- Exercise: printing top results, but stop when they get below 90:

## Solution to exercise

```
1  var vspResults = [99, 96, 93, 91, /* ... */, 41];
2
3  for (var i = 0; ( (i < vspResults.length) && (
       vspResults[i] >= 90) ); i++) {
4      console.log("Score #" + (i+1) + ": " +
           vspResults[i]);
5  }
```

# While Loops



```
while (condition) {
    // Do stuff...
}
```

- Example: print all results which are above or equal to the passing grade

```
var vspResults = [99, 96, 93, 91, /* ... */, 41];
var passingGrade = 50;

var i = 0;
while (vspResults[i] >= passingGrade) {
    console.log(vspResults[i]);
    i++;
}
```

# Do-While Loops



```
do {
    // Do stuff...
} while (condition);
```

- Example: print the first result, and then print all results which are above or equal to the passing grade

```
var vspResults = [99, 96, 93, 91, /* ... */, 41];
var passingGrade = 50;

var i = 0;
do {
    console.log(vspResults[i]);
    i++;
} while (vspResults[i] >= passingGrade);
```

# Basic Objects

# Associative Arrays



- In addition to storing items by index, an associative array can also store items by key

```
1  var vspResults = {
2      Jane:99,
3      Bob:96,
4      Kevin:93,
5      Julien:91,
6      John:41};
```

- One can access vspResults as follows:

```
1  console.log(vspResults["Jane"]); // prints 99
2  console.log(vspResults["Bob"]); // prints 96
```

# Iterating over an Associative Array



- In addition to storing items by index, an associative array can also store items by key

```
1  var vspResults = {
2      Jane:99,
3      Bob:96,
4      Kevin:93,
5      Julien:91,
6      John:41};
7
8  for (var e in vspResults) {
9      console.log(e);                // Print the name
10     console.log(vspResults[e]); // Print the score
11 }
```

- We can also use the following syntax to access an element. Word of caution: will only work for simple, non-separated identifiers!

```
1  console.log(vspResults.Jane); // prints 99
```

# String Object



- String objects store arbitrary text
- Many methods are proposed to operate on strings:
- Please refer to: https://javascript.info/string