Construction of User Interfaces (SE/ComS 319)

Jinu Susan Kabala

Department of Computer Science

Iowa State University, Fall 2021

JAVASCRIPT SPOTLIGHTS

How to add js to html file

// how to include in html file

<script> your javascript code goes in here **</script>**

// can also include from a separate file

<script src="./01_example.js"></script>

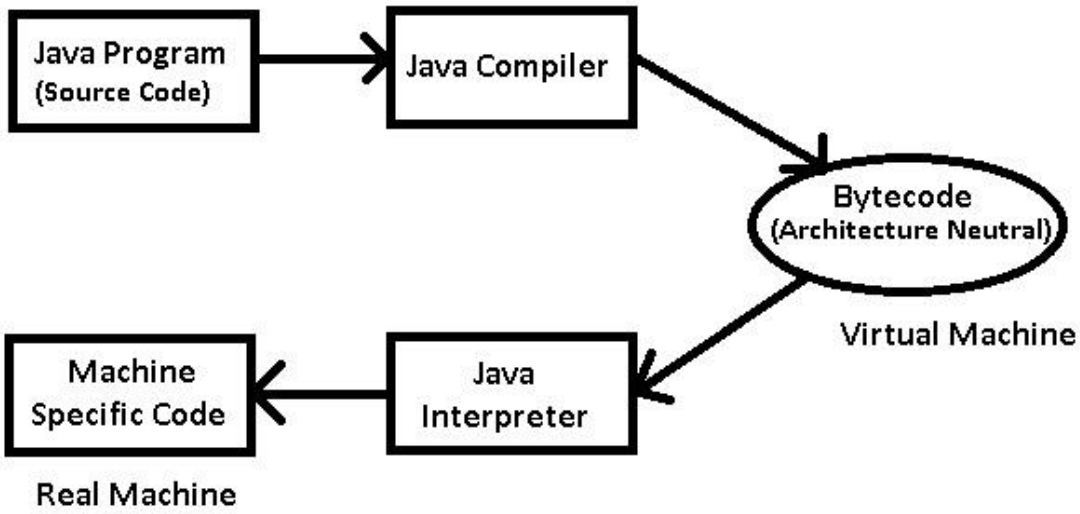
// can include from a remote site

<script src="http://…./a.js"></script>

JavaScript – Interpreter

Interpreter vs. compiler?

- JavaScript is **interpreted** at runtime by the client browser
  - Similar to Java Virtual Machine (JVM) that interprets **byte code**
- Java compiler
  - Provides an abstract machine that is programmed in Java
  - Is based on another abstract machine, i.e. the Java VM
  - Machine commands, bytecode is hidden
  - The masking is checked by the compiler



Interpreter vs. Compiler

Interpreter: (example: **JavaScript**, Python, Ruby).

- Translates program one statement at a time ➔ less amount of time to analyze the source code but the overall execution time is slower.
- No intermediate object code is generated ➔ memory efficient.
- Continues translating the program until the first error is met, in which case it stops ➔ debugging is easy.

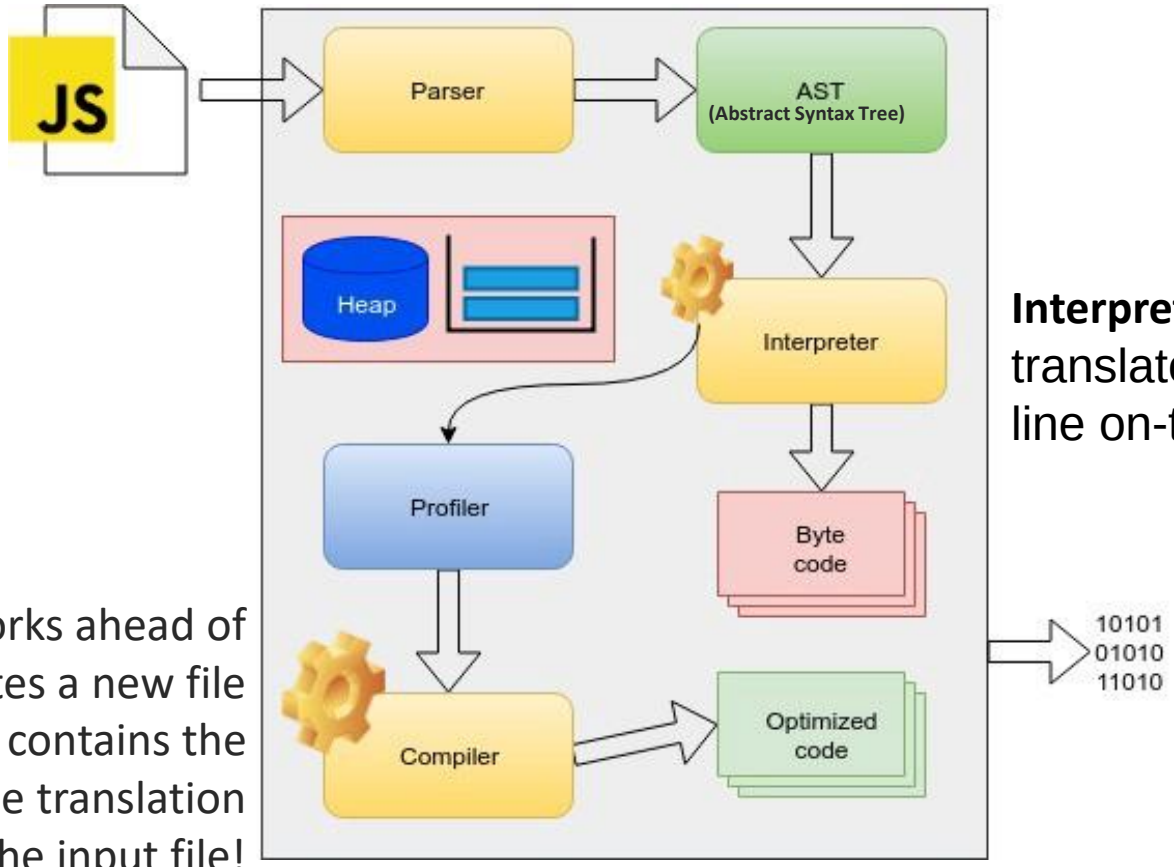
Interpreter vs. Compiler (2)

Compiler: (example: C/C++)

- Scans the entire program and translates it into machine code ➔ large amount of time to analyze the source code but the overall execution time is comparatively faster.
- Generates intermediate object code and requires linking ➔ more memory
- Error message after scanning the whole program ➔ Debugging hard

JavaScript – Interpreter (2)

- How does JavaScript work?



Interpreter reads and translates the file line by line on-the-fly!

Compiler works ahead of time and creates a new file which contains the machine code translation of the input file!

Machine code

JavaScript syntax

The JavaScript syntax is similar to C# and Java

- Operators (+, *, =, !=, &&, ++, …)
- Variables (**typeless**) ➔ JavaScript is typeless
  - Variables can store any sort of data
  - Doesn't prioritize type safety
- Conditional statements (if, else)
- Loops (for, while)
- Arrays (my_array[])
- Associative arrays (my_array['abc'])
- Functions

- **TypeScript** is a typed superset of **JavaScript** (from google)
- **TypeScript** brings you optional static type-checking
- Our focus: **JavaScript**

JavaScript basics

- **Declaring variables**: var keyword, typeless variables
  var x, y, z;

- **Basic I/O**: document.writeln( );
  window.alert( );
  window.prompt( );
  console.log();

- **Type conversion**: parseInt( );

Data types

JavaScript data types:

- Numbers (integer, floating-point)
- Boolean (true / false)
- String type – string of characters

```
var myName = "You can use both single or double quotes for strings";
```

- Arrays

```
var my_array = [1, 5.3, "aaa"];
```

- Associative arrays (hash tables)

```
var my_hash = {a:2, b:3, c:"text"};
```

Data types (2)

- Every variable can be considered as **object**
  - Arrays are objects!
- Objects use **names** to access its "members"
- Example
  - **person.firstName** returns **John**:

```
var person = {firstName:"John", lastName:"Doe", age:46};
```

Static and dynamic binding

- A binding is *static* if it first occurs before run time and remains unchanged throughout program execution
- A binding is *dynamic* if it first occurs during execution or can change during execution of the program
- **JavaScript: Dynamic Type Binding!**

Dynamic type binding in JavaScript

- Specified through an assignment statement

```
list = [2, 4.33, 6, 8];
list = 17.3;
```

- Advantage:
  - Flexibility (generic program units)
- Disadvantages:
  - High cost (dynamic type checking, interpretation, and dynamic storage allocation)
  - Type error detection by the compiler is difficult (less reliable)
  - Usually implemented using pure interpretation

String operations

The **+** operator joins strings

```
string1 = "fat ";
string2 = "cats";
alert(string1 + string2);  // fat cats
```

What is "9" + 9?

```
alert("9" + 9);  // 99
```

Converting string to number:

```
alert(parseInt("9") + 9);  // 18
```

Arrays operations and properties

Declaring new empty array:

```
var arr = new Array();
```

Declaring an array holding few elements:

```
var arr = [1, 2, 3, 4, 5];
```

Appending an element / getting the last element:

```
arr.push(3); // Add items to the end of an array
var element = arr.pop(); // Remove an item from the end of an array
```

Reading the number of elements (array length):

```
arr.length;
```

Variable declaration: var vs. let

- **let** : declare variables that are limited in scope to the block, statement, or expression on which it is used.

```
{
  let x = 2;
}
// x can NOT be used here
```

- **var** : defines a variable globally, or locally to an **entire function regardless of block scope**.

```
{
  var x = 2;
}
// x CAN be used here
```

Example:

```
var x = 10;
// Here x is 10
{
  var x = 2;
  // Here x is 2
}
// Here x is ???
```

Scope (variable access)

- **Local Scope** allow access to everything within the boundaries (inside the box)
- **Global Scope** is everything outside the boundaries (outside the box).
  - A global scope can not access a variable defined in local scope because it is enclosed from the outer world, except if you return it.

```
> function showName() {
    var name = "GeeksforGeeks";
}
showname()
console.log(name);
▶Uncaught ReferenceError: showname is not defined
    at <anonymous>:4:1
```

```
> function showName() {
    var name= "GeeksforGeeks";
    console.log(name);
}

showName();
GeeksforGeeks
```

- Limits the visibility or availability of a variable
- Separates logic in your code and improves the readability

Everything is object!

Every variable can be considered as object

- For example, strings and arrays have member functions:

```
var test = "some string";
alert(test.charAt(5)); // shows letter 's'
alert("test".charAt(1)); //shows letter 'e'
alert("test".substring(1,3)); //shows 'es'
```

```
var arr = [1,3,4];
alert (arr.length); // shows 3
arr.push(7); // appends 7 to end of array
alert (arr[3]); // shows 7
```

Sum of numbers – Example

sum-of-numbers.html

```
<html>

<head>
  <title>JavaScript Demo</title>
  <script type="text/javascript">
    function calcSum() {
      value1 =
        parseInt(document.mainForm.textBox1.value);
      value2 =
        parseInt(document.mainForm.textBox2.value);
      sum = value1 + value2;
      document.mainForm.textBoxSum.value = sum;
    }
  </script>
</head>
```

Switch statement

The `switch` statement works like in C# / Java:

```
switch (variable) {
  case 1:
    // do something
    break;
  case 'a':
    // do something else
    break;
  case 3.14:
    // another code
    break;
  default:
    // something completely different
}
```

Loops

Like in C# / Java / C++

- for loop
- while loop
- do … while loop

```
var counter;
for (counter=0; counter<4; counter++) {
  alert(counter);
}
while (counter < 5) {
  alert(++counter);
}
```

Operators: pre-increment vs. post-increment (++x vs. x++)

- **++x** (pre-increment): increment the variable; the value of the expression is the final value
- **x++** (post-increment): remember the original value, then increment the variable; the value of the expression is the original value.
  - **Example**:

```
x = 0;
y = array[x++];
// This will get array[0]

x = 0;
y = array[++x];
// This will get array[1]
```

Functions

```
function average(a, b, c)
{
    var total;
    total = a+b+c;
    return total/3;
}
```

Parameters come in here.

Declaring variables is optional. Type is never declared.

Value returned here.

Function arguments and return value

- Functions are not required to return a value
- When calling function it is not obligatory to specify all of its arguments
  - The function has access to all the arguments passed via arguments array

```
function sum() {
  var sum = 0;
  for (var i = 0; i < arguments.length; i ++)
    sum += parseInt(arguments[i]);
  return sum;
}
alert(sum(1, 2, 4));
```

Standard popup boxes

- Alert box with text and [OK] button
  - Just a message shown in a dialog box:

```
alert("Some text here");
```

- Confirmation box
  - Contains text, [OK] button and [Cancel] button:

```
confirm("Are you sure?");
```

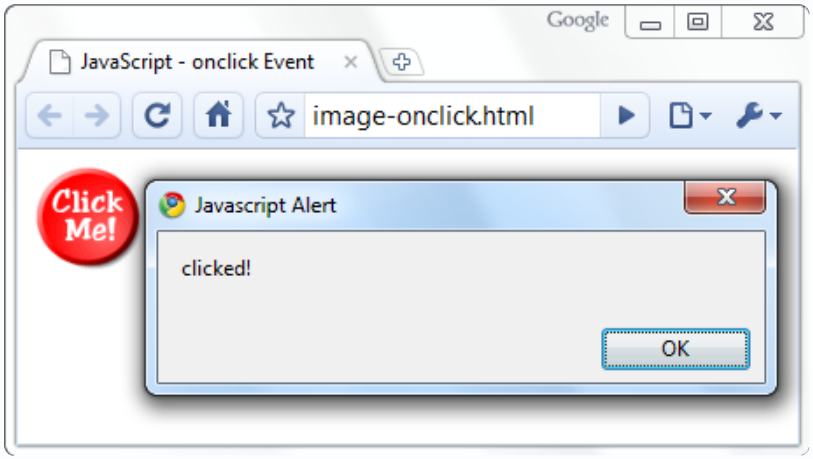
- Prompt box
  - Contains text, input field with default value:

```
prompt ("enter amount", 10);
```

Calling a JavaScript function from Event Handler – Example

```
<html>
<head>
<script type="text/javascript">
  function test (message) {
    alert(message);
  }
</script>
</head>

<body>
  <img src="logo.gif"
    onclick="test('clicked!')" />
</body>
</html>
```



While loops

```
while( expression )
   statement;
```

Executes a statement until expression becomes false

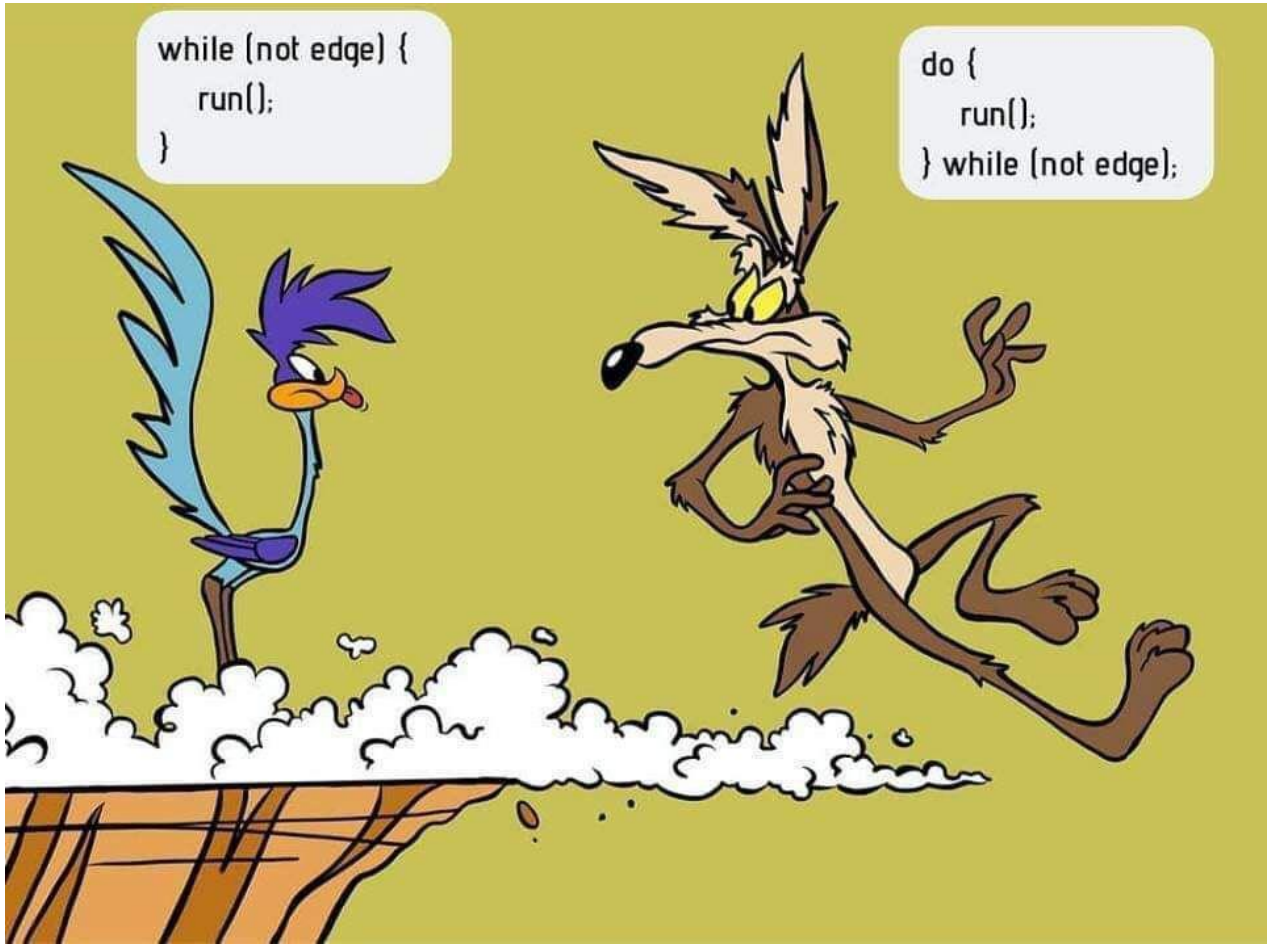
Evaluates expression before first iteration

```
do
   statement;
while( expression );
```

Evaluates expression after first iteration

Executes statement at least once

While loops – “while” vs. “do… while”



While-loop example

Print the numbers 0 to 99 to the screen

```
int i = 0;
while( i < 100 )
{
 alert(i);
 i++;
}
```

For loops (1)

General format:

```
for( expr1; expr2; expr3 )
   statement;
```

`expr1` is executed at the beginning of the loop

`expr2` is executed at the beginning of every iteration

- If it is false, the loop ends

`expr3` is executed at the end of every iteration

For loops (2)

General format:

```
for( expr1; expr2; expr3 )
   statement;
```

It is possible to omit any of the expressions

- The semicolon must stay

If `expr2` is omitted, the condition is always true

- it becomes an infinite loop

```
for( ;; ) //infinite loop
```

For loops (3)

Usual use case:

```
int i;
for( i=0; i < 100; i++ )
  alert(i);
```

Break statement

```
break;
```

Terminates the innermost loop or switch statement

Execution resumes after the loop or switch statement

```
while( 1 )
{
   n++;
   if ( n > 5 ) break;
}
```

Continue statement

```
continue;
```

Terminates the current iteration of the innermost loop

Execution resumes at the beginning of the next iteration

```
for (i=0; i<100; i++)
{
  if ( i == 57 ) continue;
  alert( i );
}
```

Print the numbers 0 to 99, but not 57

Accessing DOM

- GET the DOM element by ID or CLASS attributes

document.getElementById("xyz")

document.getElementByClassName("abc")

someDOMelement.value // this is value of the element

Accessing DOM – Example

```
<html>
<body>
<h2>JavaScript Arrays</h2>

<p>JavaScript array elements are accessed using
numeric indexes (starting from 0).</p>

<p id="demo"></p>

<script>
var cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML =
cars[0];
</script> </body> </html>
```

How to print

```
document.write()          // write to DOM
document.write("test")

console.log()             // write to console

alert()                   // popup

<p id="xyz" class="abc"> </p>
document.getElementById("xyz").innerHTML= "hi"
```

Demonstration

- Demonstration
  - JavaScript example in Browser
- Good resource for JavaScript:
  - https://www.w3schools.com/
- https://dom.spec.whatwg.org
- https://tc39.es/ecma262/
- https://v8.dev/
- https://developer.mozilla.org/en-US/docs/Web

Literature – JavaScript

- https://www.w3schools.com/