
CS472 WAP

# Key JavaScript Concepts

JavaScript for Modern Web Apps

**Maharishi International University Fairfield, Iowa** © 2020

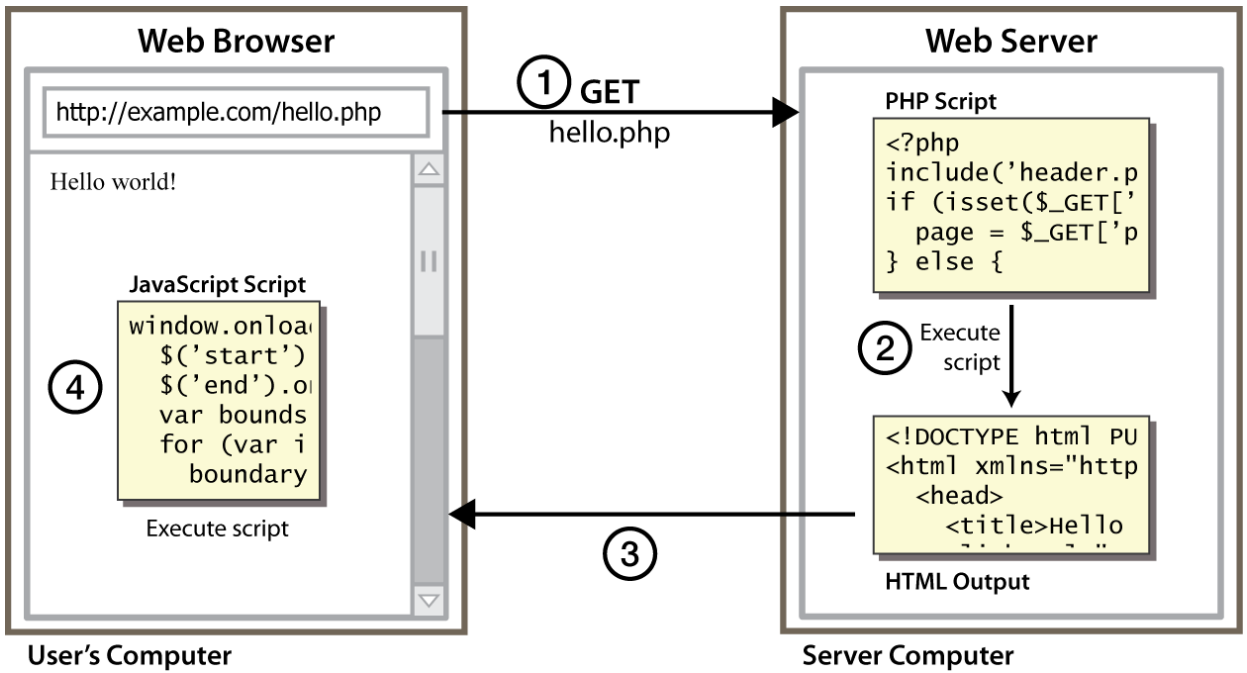
# Main Point Preview

- JavaScript is a loosely typed language.  It has types but does no compile time type checking.  Programmers must be cautious of automatic type conversions, including conversions to Boolean types.  It has a flexible and powerful array type as well as distinct types of null and undefined.

- *To be an effective JavaScript programmer one needs to understand the principles and details of the language.  If our awareness is established in the source of all the laws of nature, then our actions will spontaneously be in accord with the laws of nature for a particular environment.*

# Client-side Scripting

- client-side script: code runs in browser *after* page is sent back from server
  - often this code manipulates the page or responds to user actions



# Why use client-side programming?

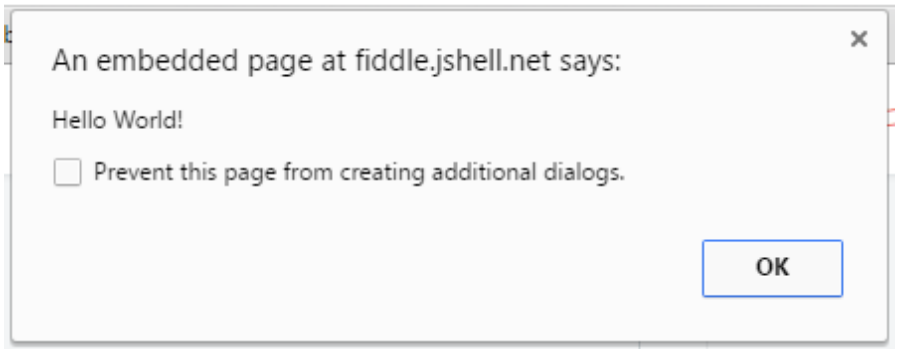
- client-side scripting (JavaScript) benefits:
  - usability: can modify a page without having to post back to the server (faster UI)
  - efficiency: can make small, quick changes to page without waiting for server
  - event-driven: can respond to user actions like clicks and key presses

# What is JavaScript?

- a lightweight programming language ("scripting language")
- used to make web pages interactive
  - insert dynamic text into HTML (ex: user name)
  - react to events (ex: page load user click)
  - get information about a user's computer (ex: browser type)
  - perform calculations on user's computer (ex: form validation)
- a web standard (but not supported identically by all browsers)
- NOT related to Java other than by name and some syntactic similarities

# A JavaScript statement: alert

```
alert("Hello World!");
```




- A JS command that pops up a dialog box with a message

# Variables and types

```
var name = expression;


var age = 32;
var weight = 127.4;
var clientName = "Connie Client";
```

- variables are declared with the var keyword (case sensitive)
  - Replaced by let and const with JS6
- types are not specified, but JS does have types ("loosely typed")
  - Number, Boolean, String, Object (Array , Function) , Null, Undefined
  - can find out a variable's type by calling typeof

# Block-scoped variable (ES6)

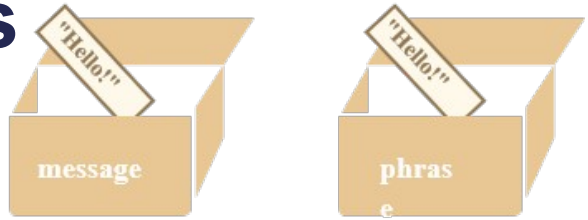
```
let callbacks = []
for (let i = 0; i <= 2; i++) {
  callbacks[i] = function() {
    return i * 2
  }
}

callbacks[0]() === 0
callbacks[1]() === 2
callbacks[2]() === 4
```

# Constants (ES6)

```
const pi = 3.1415926;
pi = 3.14; //error
const point = {x:1, y: 10};
point.y = 20; //ok
```

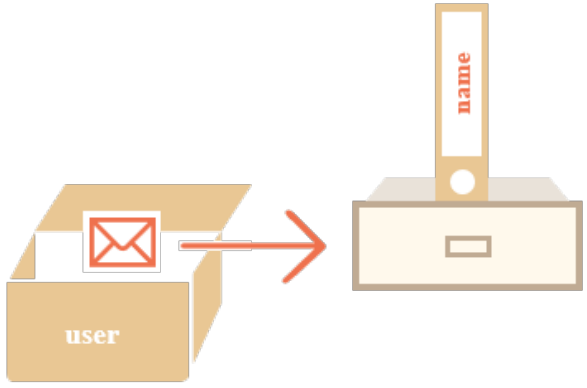
- always use const or let
  - If any doubt, use const
- also known as "immutable variables",
  - cannot be re-assigned new content.
- only makes the variable itself immutable, not its assigned content
  - object properties can be altered
  - array elements can be altered

# Variables as values versus references




- fundamental differences of objects vs primitives is that they are stored and copied “by reference”
- Primitive values: strings, numbers, booleans – are assigned/copied “as a whole value”.

```
let message = "Hello!";
let phrase = message;
```

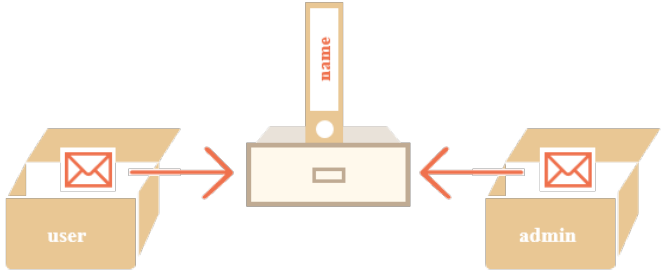



- A variable stores not the object itself, but its “address in memory”, in other words “a reference” to it.

```
let user = {
  name: "John"
};
```

- When an object variable is copied – the reference is copied, the object is not duplicated.

```
let user = { name: "John" };
let admin = user; // copy the reference
```



# Number Type

```
let enrollment = 99;
let medianGrade = 2.8;
let credits = 5 + 4 + (2 * 3);
```

- integers and real numbers are the same type (no int vs. double)
- same operators: + - * / % ++ -- = += -= *= /= %=
- similar precedence to Java
- many operators auto-convert types: "2" * 3 is 6
  - What is “2” + 3 ?

# Logical Operators

- >, <, >=, <=, &&, ||, ! ==, !=, ===, *!==*
- most logical operators automatically convert types:
  - 5 < "7" is true
  - 42 == 42.0 is true
  - "5.0" == 5 is true
- === and !== are *strict equality* tests
  - checks both type and value
  - "5.0" === 5 is false
- Always use *strict* equality

# Comments (same as Java)

```
// single-line comment
/* multi-line comment */
```

- identical to Java's comment syntax
- recall: 4 comment syntaxes
  - HTML: <!-- *comment* -->
  - CSS/JS/PHP/Java: /* *comment* */
  - Java/JS/PHP: // *comment*
  - Python: # *comment*

# String Type

```
const s = "Connie Client";
let fName = s.substring(0, s.indexOf(" ")); // "Connie"
let len = s.length; // 13
let s2 = 'Melvin Merchant'; // can use "" or ' '
```

- methods: charAt, charCodeAt, fromCharCode, indexOf, lastIndexOf, replace, split, substring, toLowerCase, toUpperCase
  - charAt returns a one-letter String (there is no char type)
- length property (not a method as in Java)
- concatenation with + : 1 + 1 is 2, but "1" + 1 is "11"

# More about String

- escape sequences behave as in Java: \' \" \& \n \t \\
- to convert between numbers and Strings:

```
let count = 10;
let s1 = "" + count; // "10"
let s2 = count + " bananas, ah ah ah!"; // "10 bananas, ah ah
  ah!"
const n1 = parseInt("42 is the answer"); // 42
const n2 = parseFloat("booyah"); // NaN
```

- to access characters of a String, use [*index*] or charAt:

```
const firstLetter = s[0];
let firstLetter = s.charAt(0);
let lastLetter = s.charAt(s.length - 1);
```

# Boolean Type

```
const iLikeWebApps = true;
const ieIsGood = "IE6" > 0; // false
if ("web dev is great") { /* true */ }
if (0) { /* false */ }
```

- any value can be used as a Boolean
  - "falsey" values: false, 0, 0.0, NaN, empty String(""), null, and undefined
  - "truthy" values: anything else, include objects
- !! Idiom – gives boolean value of any variable
  - const x=5;
  - console.log(!x); //false
  - console.log(x);  //5
  - console.log(!!x);  //true

# Special values: null and undefined

```
let ned = null;
const benson = 9;
let caroline;
alert(fred);  // reference error, fred not declared
// at this point in the code,
// ned is null
// benson's 9
// caroline is undefined
```

- undefined : has been declared, but no value assigned
  - e.g., caroline variable above
  - vars that are "hoisted" to beginning of a function
- null : exists, and was specifically assigned a value of null
- reference error when try to evaluate a variable that has not been declared
  - reference error different from undefined
    - E.g., const y = x;
    - Uncaught ReferenceError: x is not defined  //if x has never been declared
  - undefined means declared, but no value assigned

# if/else statement (same as Java)

```
if (condition) {
  statements;
} else if (condition) {
  statements;
} else {
  statements;
}
```

- identical structure to Java's if/else statement
- JavaScript allows almost anything as a *condition*
  - *JS idioms*
  - //initialize a variable if not set yet
    - if (!a) { a = 10;}
  - //only use a variable if it has a value
    - if (b) { console.log(b); }

# for loop (same as Java)

```
for (initialization; condition; update){
 statements;
}

let sum = 0;
for (let i = 0; i < 100; i++) { //could we use const?
 sum = sum + i;
}

const s1 = "hello";
let s2 = "";
for (let i = 0; i < s1.length; i++) {
 s2 += s1[i] + s1[i];
} // s2 stores "hheellllloo"
```

# while loops (same as Java)

```
while (condition) {
statements;
}

do {
statements;
} while (condition);
```

- break and continue keywords also behave as in Java

# Arrays

```
let name = []; // empty array
let name = [value, value, ..., value]; // pre-filled
name[index] = value; // store element

const ducks = ["Huey", "Dewey", "Louie"];
let stooges = []; // stooges.length is 0
stooges[0] = "Larry"; // stooges.length is 1
stooges[1] = "Moe"; // stooges.length is 2
stooges[4] = "Curly"; // stooges.length is 5
stooges[4] = "Shemp"; // stooges.length is 5
```

- two ways to initialize an array
- length *property* (grows as needed when elements are added)

# Array methods

```
let a = ["Stef", "Jason"]; // Stef, Jason
a.push("Brian"); // Stef, Jason, Brian
a.unshift("Kelly"); // Kelly, Stef, Jason, Brian
a.pop(); // Kelly, Stef, Jason
a.shift(); // Stef, Jason
a.sort(); // Jason, Stef
```

- array serves as many data structures: list, queue, stack, ...
- methods: concat, join, pop, push, reverse, shift, slice, sort, splice, toString, unshift
  - push and pop add/remove from back (efficient)
  - unshift and shift add/remove from front (not efficient)
  - shift and pop return the element that is removed

# Array methods: map, filter, reduce

```
//functional programming:  map, filter, reduce can replace many loops

const a = [1,3,5,3,3];

//return new array with elements translated/mapped to another set of values
const b = a.map(function(elem, i, array) {
  return elem + 3;})// [4,6,8,6,6]

//return new array containing select elements based on a condition
const c = a.filter(function(elem, i, array){
  return elem !== 3;});//[1,5]

//find first element or index of first element satisfying condition
const d = a.find(function(elem) {return elem > 1;}); //3
const e = a.findIndex(function(elem) {return elem > 1;}); //1

//accumulate a single value based on elements across the array
const f = a.reduce(function(accumulator, elem){
  return accumulator + elem;}, 0); //0 is initial value, 15 is sum
```


MAHARISHI
HIGHER EDUCATION
FOR HIGHER CONSCIOUSNESS
INTERNATIONAL UNIVERSITY