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?

PHP already allows us to create dynamic web pages. Why also use client-side scripting?

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

Server-side programming (PHP) benefits:
- **security**: has access to server's private data; client can't see source code
- **compatibility**: not subject to browser compatibility issues
- **power**: can write files, open connections to servers, connect to databases, ...
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
JavaScript vs. Java

- **interpreted**, not compiled
- more relaxed syntax and rules
  - fewer and "looser" data types
  - variables don't need to be declared
  - errors often silent (few exceptions)
- key construct is the **function** rather than the class
  - "first-class" functions are used in many situations
- contained within a web page and integrates with its HTML/CSS content
JavaScript vs. PHP

• similarities:
  • both are interpreted, not compiled
  • both are relaxed about syntax, rules, and types
  • both are case-sensitive
  • both have built-in regular expressions for powerful text processing

• differences:
  • JS is more object-oriented: noun.verb(), less procedural: verb(noun)
  • JS focuses on UIs and interacting with a document; PHP on HTML output and files/forms
  • JS code runs on the client's browser; PHP code runs on the web server
Linking to a JavaScript file: script

- **script** tag should be placed in HTML page's head
- script code is stored in a separate `.js` file
- JS code can be placed directly in the HTML file's **body** or **head** (like CSS)
  - but this is bad style (should separate content, presentation, and behavior)
A JavaScript statement: alert

- a JS command that pops up a dialog box with a message
Variables and types

| var name = expression; | JS |
| var age = 32; |
| var weight = 127.4; |
| var clientName = "Connie Client"; | JS |

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

```
var enrollment = 99;
var medianGrade = 2.8;
var 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
String type

```javascript
var s = "Connie Client";
var fName = s.substring(0, s.indexOf(" ")); // "Connie"
var len = s.length; // 13
var 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 \n`

• to convert between numbers and Strings:

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

• to access characters of a String, use `[index]` or `charAt`:

```javascript
var firstLetter = s[0];
var firstLetter = s.charAt(0);
var lastLetter = s.charAt(s.length - 1);
```
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:/ * comment */
  - Java/JS/PHP:// comment
  - PHP:# comment
## for loop (same as Java)

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

```javascript
var sum = 0;
for (var i = 0; i < 100; i++) {
  sum = sum + i;
}
```

```javascript
var s1 = "hello";
var s2 = "";
for (var i = 0; i < s.length; i++) {
  s2 += s1[i] + s1[i];
}
// s2 stores "hheelllloo"
```
Math object

```javascript
var rand1to10 = Math.floor(Math.random() * 10 + 1);
var three = Math.floor(Math.PI);
```

- methods: `abs`, `ceil`, `cos`, `floor`, `log`, `max`, `min`, `pow`, `random`, `round`, `sin`, `sqrt`, `tan`
- properties: `E`, `PI`
Logical operators

- Relational: \( > \), \( < \), \( \geq \), \( \leq \)
- Logical: \( \&\& \), \( || \), \( ! \)
- Equality: \( == \), \( != \), \( === \), \( !== \)
  - most logical operators automatically convert types. These are all true:
    - \( 5 < "7" \)
    - \( 42 == 42.0 \)
    - \( "5.0" == 5 \)
  - The \( === \) and \( !== \) are strict equality tests; checks both type and value:
    - \( "5.0" === 5 \) is false
if/else statement (same as Java)

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

- identical structure to Java's `if/else` statement
- JavaScript allows almost anything as a `condition`
Boolean type

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

• any value can be used as a Boolean
  • "falsey" values: 0, 0.0, NaN, "", null, and undefined
  • "truthy" values: anything else
• converting a value into a Boolean explicitly:
  • `var boolValue = Boolean(otherValue);`
  • `var boolValue = !!otherValue;`
while loops (same as Java)

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

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

- `break` and `continue` keywords also behave as in Java but do not use them in this class!
# Arrays

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

```javascript
var 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
  - `unshift` and `shift` add / remove from front
  - `shift` and `pop` return the element that is removed
Splitting strings: split and join

```javascript
var s = "the quick brown fox";
var a = s.split(" "); // ["the", "quick", "brown", "fox"]
a.reverse(); // ["fox", "brown", "quick", "the"]
s = a.join("!"); // "fox!brown!quick!the"
```

- split breaks apart a string into an array using a delimiter
- can also be used with regular expressions surrounded by `/`:
  ```javascript
  var a = s.split(/[ \t]+/);
  ```
- join merges an array into a single string, placing a delimiter between them
Defining functions

```javascript
function name() {
    statement;
    statement;
    ...
    statement;
}
```

```javascript
function myFunction() {
    alert("Hello!");
    alert("How are you?");
}
```

- the above could be the contents of `example.js` linked to our HTML page
- statements placed into functions can be evaluated in response to user events
Special values: null and undefined

```javascript
var ned = null;
var benson = 9;
var caroline;

// at this point in the code,
// ned is null
// benson's 9
// caroline is undefined
```

- **undefined**: has not been declared, does not exist
- **null**: exists, but was specifically assigned an empty or null value
- **Why does JavaScript have both of these?**
Event-driven programming

- JS programs have no `main`; they respond to user actions called **events**
- **event-driven programming**: writing programs driven by user events
## Event handlers

<table>
<thead>
<tr>
<th>HTML</th>
<th>HTML</th>
<th>HTML</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;element attributes onclick=&quot;function();&quot;&gt;...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;div onclick=&quot;myFunction();&quot;&gt;Click me!&lt;/div&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Click me!</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- JavaScript functions can be set as **event handlers**
  - when you interact with the element, the function will execute
- **onclick** is just one of many event HTML attributes we'll use
Buttons: `<button>`

*the canonical clickable UI control (inline)*

```html
<button onclick="myFunction();">Click me!</button>
```

- button's text appears inside tag; can also contain images
- To make a responsive button or other UI control:
  1. choose the control (e.g. button) and event (e.g. mouse click) of interest
  2. write a JavaScript function to run when the event occurs
  3. attach the function to the event on the control
Accessing an element: document.getElementById

```javascript
var name = document.getElementById("id");
```

```html
<img id="icon01" src="images/octopus.jpg" alt="an animal" />
<button onclick="changeImage();">Click me!</button>
```

```javascript
function changeImage() {
    var octopusImage = document.getElementById("icon01");
    octopusImage.src = "images/kitty.gif";
}
```

- `document.getElementById` returns the DOM object for an element with a given `id`