ECMAScript

Formally JavaScript

ECMAScript scripts in HTML:

3 ways to incorporate ECMAScript:
1) Inside of a script section:

```
<script>
   [ES code executed as the page loads.]
</script>
```

- Code is executed as the page loads, very much like PHP code is, so any ES code outside of a function is executed immediately as the page is processed.

2) An external script:

```
<script src='file.js'></script>
```

- The script is loaded and executed as if it were written inside of a script section, thus the script file should not include the <script> tags.

3) In an event handler attribute:

```
<div onclick='[ECMAScript code]'> ... </div>
```

- Installs an event handler (onclick) for the element. The ECMAScript code inside the string is executed as if in a <script> section, when that event occurs. The event object **event** (or **window.event**, as **window** is the global object) only exists at the time of the event so should be passed as a parameter to any functions where it is needed. The keyword this also can be used to refer to the element object that triggered the event. Ex:

```
<div onclick='foo(this,event);'></div>
```

Event-driven programming:

- A programming system in which the execution of the program is determined by "events", such as user input or I/O messages.
- There exists a "main loop" which listens for events and responds by calling "callback" functions (event handlers) created by the programmer to handle the events.
- Callbacks are defined in the HTML via the on* attributes placed in elements, or can be explicitly defined to an object in ECMAScript script sections using

```
addEventListener(eventname, callback)
```

or setting the object property of the name *eventname* to a callback function: i.e.:

```
object.onmouseover = function() {  };
object.addEventListener("onmouseover", function() {  });
```

The ECMAScript console:

- Most modern browsers include a ECMAScript "console" and debugger. In Firefox and Chromium this is reached with the Ctrl-Shift-i or Ctrl-Shift-j keys.

ECMAScript events:

Mouse events:

onclick	A element has been clicked on
onmousedown	Mouse button is pressed down
onmousemove	Mouse pointer moves over an element
onmouseup	Mouse button is released

Keyboard events:

onkeypress	User pressed a key

Frame/object events:

onload	Page or object has finished loading.
onresize	View has been re-sized.

Form events:

onblur	Element loses focus
onchange	Element has been changed
onfocus	Element gains focus
oninput	Element gets input
onsubmit	Form is submitted

ECMAScript Data types:

- Types are dynamic

```
var x = 1;              // x is a number.
x = "10";               // x is now a string.
```

- Strings:

```
var x = "abc";          // Double quoted
var x = 'xyz';          // Single quoted
```

- Numbers:

```
var x = 10;
var x = 10.5;
var x = 10e5;
var x = 0xFF;           // hexadecimal
var x = 0777;           // Octal
var x = 0b11010;        // Binary
```

  - Always 64 bit floating point values (equivalent to a double in C).
  - A string that contains a number may be automatically converted to a number it is used in a numeric operation. Ex: `var x = 10 / "2";`

- Boolean:

```
var x = true;
var x = false;
```

- Arrays:

Dynamically sized:

```
var x = new Array();    // array via object creation method
var x = [];             // Literal array creation (preferred method)
var x[0] = 1;
var x[1] = 2;
…
var x = new Array(1, 2, "text");
var x = [ 1, 2, "text" ];
```

  - Arrays are really just objects where the numeric indices are names with associated values, thus all arrays may be sparse associative arrays.

- Objects:
  - Are lists of name/value pairs (i.e. a dictionary or map)

```
var car = {
  model : "Kia",
  make  : "Optima",
  year  : 2012
}
// The following are equivalent:
make = car.make;
make = car["make"];
```

  - If the name is valid identifier you can use the dot (.) accessor instead of array ([]) accessor.

- Undefined and Null:
  - **undefined** is the value of a variable with no value.
  - **null** empties a variable, but is otherwise useless.

Variables and Scope:

- Globals are variables created outside of functions or variables that are instantiated inside functions where 'var' is not prefixed. Globals do not have to use 'var' to declare them.

  - Globals created with var are not delete-able, those created w/o var are delete-able.

  - Globals and all top level functions belong to the "window" object, so may "collide" with existing names in the window object. It's a good idea to preface them with a single underscore (_) to make them unique to your scripts.

- Local variables must use 'var' when declaring them. The scope is local to the function in which it is declared, nested functions have their own scope.

- Local variables are visible to the entire function, even before they're initialized, so:

```
  var x = "global";
  function f() {
    console.log(x);       // prints undefined (x is local but not yet defined)
    var x = "local";            // defines the local variable x.
    console.log(x);       // prints local
    console.log(window.x);// Accesses the global x via the window object.
  }
```

- The keyword **let** is like **var**, but gives a variable block-local scope.

Statements:

- Expression statements: Any expression, e.g. an assignment, function-call, etc.

- Compound and Empty Statements:
  - A sequence of statements enclosed in {}'s, i.e. a block.
  - A single '**;**' preceded by nothing indicates an empty statement.

if / else if:

```
 if (expression)
   statement1
 else if (expression)
   statement2
 else
   statement3
```

switch:

```
 switch(expression) {
   statements
 }
 Ex:
 switch(n) {
  case 1:                 // Cases may match string constants, not just integers.
    break;
  case 2:
    break;
  default:
    break;
 }
```

while:

```
 while(expression) statement
```

do/while:

```
 do
   statement              // statement should always be enclosed in a block for readability.
 while (expression);
```

for:

```
 for(initialize; test-expression; increment-expression) statement

 Almost, but not quite like:
  initialize
  while (test-expression) {
    statement
    increment-expression
  }
```

for/in:

```
 for (variable in object|array) statement
```

- Iterates over each enumerable property of an object, placing the names of the property in the *variable*. Typically in the order in which they are defined in the object, but order is not specified.

for/of:

```
 for (variable of object|array) statement
```

- New in ES6, iterates over the values of an object or array rather than the names.

Jumps:
 - Labeled statements:
   identifier: *statement*
   - Gives a name to this statement which can be referred to by a continue or break.

 - **continue** [*identifier*];
   - Causes the innermost enclosing loop to continue immediately to the next iteration.

 - **break** [*identifier*];
   - Causes the innermost enclosing loop to exit immediately.

 - **return** *expression*;

 - **throw** *expression*;
   - Throws an exception, which may be handled in a **try/catch/finally** section.

try/catch/finally:
```
  try {
   - Code to be executed.
  }
  catch (e) {
   - Code to be executed in the event of an exception in the try section. 'e' refers to
     the exception object or other value that was thrown.
   - May handle the exception, ignore it, or re-throw the exception with throw.
  }
  finally {
   - Contains code that is always executed regardless of what happens in the try block.
  }

  initialize
  while (test-expression) {
    try { statement; }
    finally { increment-expression; }
  }
```

Number properties and methods:
 - **Number** is the numeric object constructor.
```
   var x = Number(10);     → Creates a numeric object with 10 as it's value.

   Number.NaN              - Not a Number property.
```
 - Useful numeric methods to convert to various kinds of strings.
```
   toExponential(n)        - Exponential with n digits of precision.
   toPrecision(n)          - n digits of precision
   toFixed(n)              - Fixed point.
   toString(base)          - Convert to string of base, default = base 10.

   var x = 10;
   x.toString();
   Number(10).toString();
   - etc.
```
 - Other useful functions:
```
   parseInt(n)             - Parse a integer string into a number.
   parseFloat(n)           - Parse a float string into a number.
   isNaN(n)                - Test if number is Not a Number
```

String Properties and Methods:
 - "String" is the object constructor for strings.
 - Most methods and properties can be used on string constants as well:
```
  "abc".length == 3
  "abc"[1] == 'b'
```

Properties:
```
  length                   - The length of the string in characters.
```

Methods:
```
  charAt(n)                - Same as str[n]
  charCodeAt(n)            - ASCII value of character at location n.
```

```
  String.fromCharCode(n)   - Converts an ASCII value to a character.
  search(re)               - Search for regex in string, returns index if found or -1 if not.
  split(delim)             - Split a string into an array of strings.
  trim()                   - Removes white-space from both ends of the string.
```

Array Properties and Methods:

- **Array** is the object constructor for an array, however just use the array literal syntax ([]).

```
Properties:
  length                   - Length of the array in elements.

Methods:
  join(delim)              - Joins the elements of the array together as a string separated
                             by delim (',' is default delim if not specified)
  toString()               - Like join(',')
  pop()                    - Removes the last element of an array.
  push(x)                  - Adds a new element x to the end of an array.
  shift()                  - Removes the first element of an array, shifting everything else
                             up one place.
  unshift(x)               - shifts everything down one place and inserts x as the new first
                             element.
  indexOf(x)               - Search array for x and return it index if found, -1 if not.
  sort([cmp])              - Sorts an array (by default alphabetically), optionally with a
                             comparison function cmp.
                             Ex: x.sort(function(a,b) { return a-b; })
  reverse()                - Reverses the elements of an array.
  concat(a[,b..])          - Concatenates an array or arrays onto the current array.
```

Event Object:

```
  preventDefault()
```

- To cancel the event if it is cancel-able, meaning that any default action normally taken by the implementation (the browser) as a result of the event will not occur.

```
  stopPropagation()
```

- To prevent further propagation of an event during event flow

```
  target                   - The element that triggered the event.
  CurrentTarget            - The element whose event listener triggered the event.
```

MouseEvent / KeyboardEvent Object:

```
  Properties:       Returns:
  button            - Which mouse button was clicked when an event was triggered (0=left, 1=middle,
                      2=right)
  detail            - # of times button was clicked (0=mousedown,1=click,2=dblclick)
  clientX           - The horizontal coordinate of the mouse pointer, relative to the current
                      window, when an event was triggered
  clientY           - The vertical coordinate of the mouse pointer, relative to the current window,
                      when an event was triggered
  charCode          - Unicode encode of the character that was generated.
  KeyCode           - Virtual keycode of the key that was pressed.
  altKey, metaKey, shiftKey, ctrlKey
```

- Whether or not the alt/meta/shit/ctrl key was pressed when an event was triggered

The Document & Element objects:

- "**document**" is the object representation of your HTML document after it's been loaded into the browser and represents the base of a tree of element "node" objects.
- It provides methods and properties to access all node elements from within ECMAScript.
- Useful document properties & methods: * = remember these

```
  cookie            - Name/value pairs of all cookies for this document.
* body              - The "body" section of the document.
* forms             - Collection all forms within a document, which can be accessed by name
                      (i.e. document.form.<name>)
  images            - Collection of all images.

* createElement(x)  - Creates an element node of type x
  createTextNode()  - Creates a text node
* getElementById(id) - Element that has the ID attribute set.
  getElementsByClassName(x) - A NodeList of elements in the specified class.

  write()           - Writes into a document stream.
```

- Useful node elements properties and methods:

```
  id                - Sets/returns the elements ID.
* classNames        - Sets/returns the value of the class attribute
* style             - Sets/returns the value of the style attribute

  clientHeight      - Height of the element (including padding)
  clientWidth       - Width of the element (including padding)
* offsetHeight      - Height including padding, border and scroll-bar
* offsetWidth       - Width including padding, border and scroll-bar
* offsetTop         - Vertical offset position of this element
* offsetLeft        - Horizontal offset position of this element
* offsetParent      - Container parent of this element.

  childNodes        - A collection of child nodes for this element.
* firstChild        - First child node of an element
* lastChild         - Last child node of an element.
* nextSibling       - Next element at the same level as this element.
* previousSibling   - Previous element at the same level.
* parentNode        - Parent of this element.

* innerHTML         - Sets/returns the content of the element
  textContent       - Sets/returns the textual content
* appendChild(e)    - Appends a child element (e) to this node.
  insertBefore(e,f) - Inserts a child elem (e) before a specified existing child (f)
  removeChild(e)    - Removes a child node (e)

  focus()           - Sets the focus on this element.
  blur()            - Removes focus from this element.

  hasAttribute(a)   - True if the element has the specified attribute
* getAttribute(a)   - Gets the attribute value for a specific attribute
* setAttribute(a,v) - Sets the attribute on the node.
```

Example:

```
// Finds the element with the ID attribute set to 'x':
var e = document.getElementById("x");
// Sets the text color to blue:
e.style.color = "blue";
```

Example:

```
<ul id='list'></ul>
<script>
// Get the above ul element by its id:
var ul = document.getElementById("list");

for(var i=0; i<5; i++) {
  // Create a new list (li) element:
  var li = document.createElement("li");
  // Set the contents of the new list element:
  li.innerHTML = "list element " + i;
  // Add it the ul element:
  ul.appendChild(li);
}
</script>
```