Persistent

JavaScript: Language Basics

Persistent Interactive | Persistent University

Key learning points :

- Overview of JavaScript
  - Evolution & History
  - Features & Characteristics
  - ECMAScript standard and versions
  - Understanding script tag
  - Inclusion of JavaScript in web pages
- Basic concepts involving :-
  - Define Variables and types and scopes
  - Data Types & Operators
  - Control & Loop statements
  - Arrays & Strings

JavaScript History

- Launched by Netscape Navigator in 1995
- Brendan Eich, worked on a scripting language called Mocha, later called LiveScript and finally named as JavaScript
- Microsoft introduced Internet Explorer with JavaScript implementation called Jscript
- Browser based language – two flavors become available with JavaScript and Jscript.

Overview of JavaScript

- JavaScript® (often shortened to JS) is :-
  - lightweight
  - object-oriented language with first-class functions
  - scripting language for web pages

- Most popular client-side scripting language

- Makes web pages interactive and dynamic

- Helps to change HTML Content and HTML styles at runtime

Overview of JavaScript continued ..

- Supports Event driven programming

- With the recent growth of numerous JavaScript libraries, it is easier to :-
  - navigate a document
  - select DOM elements
  - create animations
  - handle events
  - develop Ajax applications

- JavaScript is the only cross-platform, client-side programming language that is both free and universally adopted.

JavaScript Execution

- Supported and executed by all modern browsers

- Follows load and go execution plan
  - Writing
  - Loading
  - Executing

- Html document gets parsed from top to bottom

- External JavaScript files are preferred over embedded JavaScript code
  - although may take more time to load but separate JavaScript code from view layer

ECMAScript

- In 1997, JavaScript got standardized with ECMAScript to produce the same results on all platforms supported by the browser

- Technical Committee #39(TC-39) came out with ECMAScript-262, standard defining scripting language named ECMAScript

- JavaScript follows from ECMAScript its:
  - Syntax & Statements
  - Types & Operators
  - Keywords & Reserved Words
  - Objects

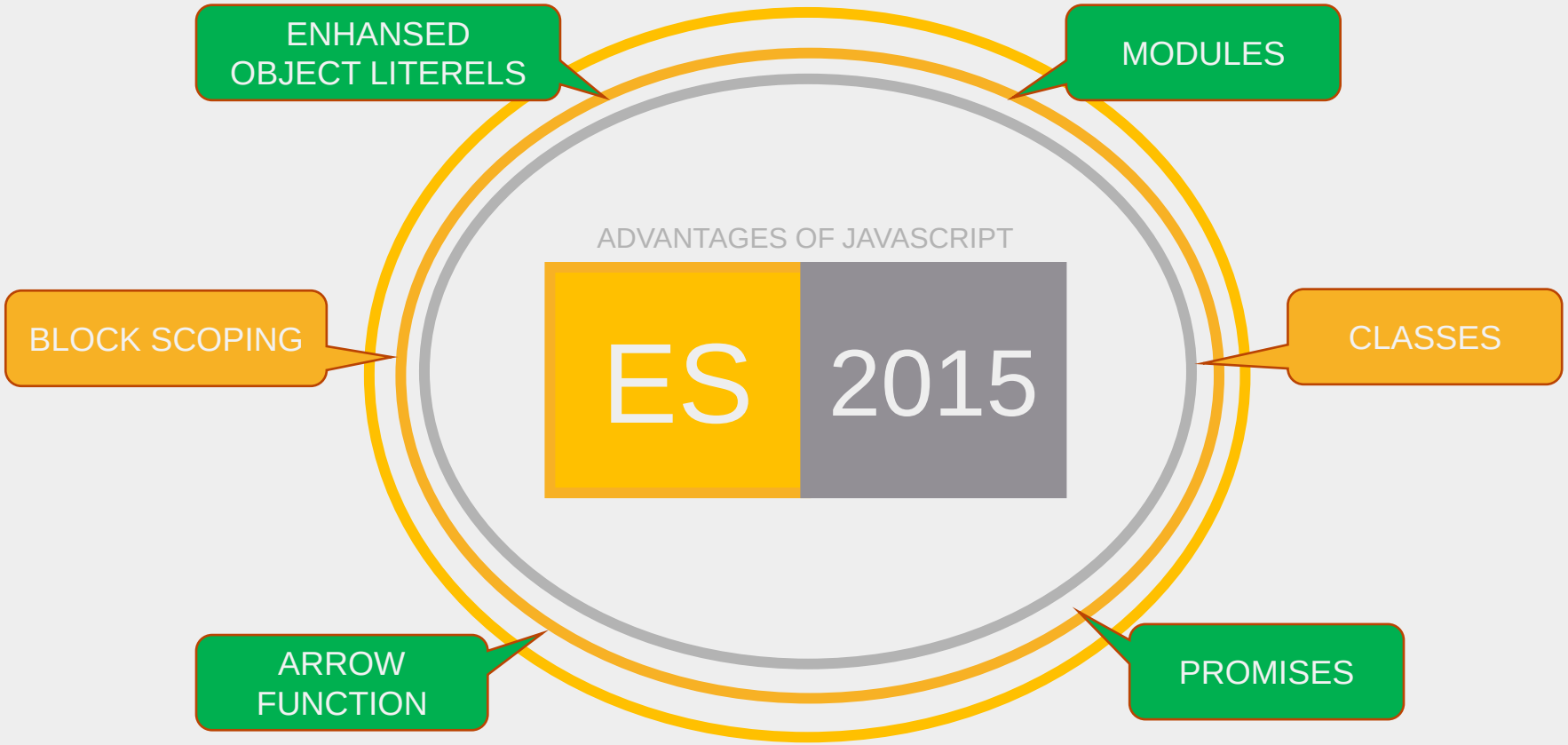
ECMAScript versions

- **ECMAScript 1**-First version of JavaScript language standardized in June 1997.
- **ECMAScript 2-** Developed in June 1998 with minor changes to keep the spec in sync with a separate ISO standard for JS.
- **ECMAScript 3-** Introduced in December 1999, with many features that have become popular parts of the language, like regular expressions, new control statements and many more.
- **ECMAScript 4-**Designed by Adobe,Mozilla,Opera and Google and was a massive upgrade with features like classes, interfaces and meta level methods etc., but never released.
- **ECMAScript 3.1-** Designed by Microsoft and Yahoo. It was an incremental upgrade of ES3 and subset of ES4 with some minor changes. This eventually became ECMAScript 5.
- **ECMAScript Harmony-**In July 2008, Brendan Eich, proposed an agreement between ES3.1 and ES4 that contains special features of both and called that ECMAScript Harmony.
- **ECMAScript 5-**Introduced in December2009, with several enhancements in the standard library and updated language semantics via strict mode. This standard has been fairly implemented in almost all of the browsers.
- **ECMAScript 5.1-**In June 2011,minor corrections were made to ES5 and that introduced as ECMAScript 5.1.

ECMAScript versions continued……..

- **ECMAScript 6-**It got standardized in 2015. This standard has been partially implemented in most of the modern browsers. Includes many new features like improved modularity, improved functions and control flow and a lot more.
- **ECMAScript 7**-It got standardized in 2016. It included 2 more features in ES6 as Array.prototype.includes() and Exponentiation operator.
- **ECMAScript 8-** It got standardized in Jan 2017. It have new features as Async Functions and Shared memory and atomics with minor new features as String.padding, Object.entries etc
- **ECMAScript 9-** It got standardized in 2018. It have new features as Asynchronous Iteration and Rest/Spread properties new Regular Expression features.

Advantage of Using ES6 Over ES5



Advantages of Using ES6 over ES5

- Arrows: are a function shorthand that share the same lexical this as their surrounding code.
- Classes: are having a single conducive declarative that boosts interoperability and easier to use.
- Enhanced Object Literals: supports setting the prototype at construction, defining methods, making super calls, computing property names with expressions.
- Modules: prevents need for global and supports standardized protocol for sharing libraries.
- Block Scoping: can be achieved using let keyword in ES6 which ES5 was lacking with.
- Promises: ES6 is a language that promises to handle its results and errors. They have improved readability via method chaining in comparison to callbacks.

<Script> tag

- A web page can have multiple script tags

- Internal and External scripts can not combine using same <script>

- Attributes of <script> tag :-
  - type
  - src
  - language
  - async
  - defer
  - charset

How to include JavaScript in HTML ?

- JavaScript code has to be placed inside the <script> tags.

- <script> tag can be added in :-
  - <head> tag
  - <body> tag

Example of adding JavaScript code in head

```
<head>
        <script type="text/javascript">
                ...
        </script>
</head>
```

Referencing External JavaScript File

- We can write JavaScript code in an external file with extension .js and include that file in the html page.

Pointing to a local js file

```
<head>
        <script  type="text/javascript"

                 src="scripts/myOwnJavaScript.js">

        </script>
</head>
```

Using a JavaScript File from web url

- We can also include an external file using web url.

Pointing to a web js file

```
<head>
        <script type="text/javascript"

        src="http://somesite/myOwnJavaScript.js">

        </script>
</head>
```

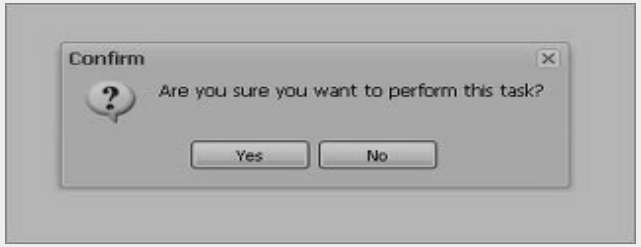
JavaScript Popup Boxes

- Alert box

```
alert(“Hello!. Sample alert”)
```

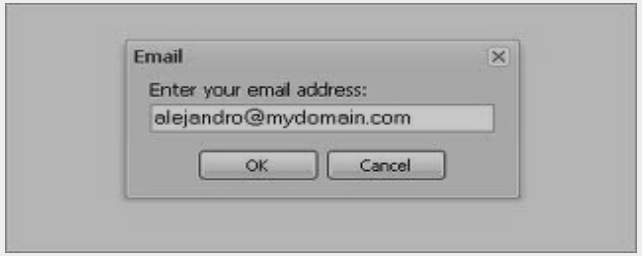
- Confirm box

```
confirm(“You sure about this task?”)
```



- Prompt box

```
prompt(“Enter email addresss");
```



Persistent

JavaScript Variables

- Variables are declared with or without the ’var’ keyword
- Datatype is internally assigned based on the value

```
var strname = “jack”;

strname = “jack”;

var age1=34;

var age2=“34”;
```

Declared using “var” keyword

Declared without using “var” keyword

age1: Number
age2: String

Persistent

Variable Naming Conventions

- Variable names have to start with a
  - letter
  - underscore(_)
  - dollar($).

- Variable names are case sensitive.

```
var firstName = “jack”;

var Lastname = “Helms”;

var _age = 34;

var $phone = 245678;
```

Local and Global Variables

Local

- Declared inside a function with ‘var’ keyword.
- Accessible only within the function

Global

- Declared outside a function or inside a function but without ‘var’ keyword.
- Accessible across all functions.

Variable Declarations

- Can declare variable using ‘var’ keyword

- Without declaration variables :-
  - get hoisted
  - become global properties outside the function declaration
  - ECMAScript strict mode throws an error in this scenario

- Stays in global context if not defined inside any function – no block scoping

- Can access global variables declared in one window or frame from another window or frame by specifying the window or frame name

Data Types : Primitive

- Number
  - double precision 64-bit format
- Boolean
- String
  - Unicode characters (16 bit)
- Undefined

39, 12.25, -220, -34.54, 23E4

true, false

“java” , ‘python’

var a;

Persistent

Data Types continued : Object

- Object
- Arrays
  - data type is Object
- Null
  - data type is Object

```
{
    'name' : 'Anand',

    'company' : 'psl'
}

[1, 2, 3]

["java", "c++", "dot net"]

var object1 = null;
```


Persistent

Operators

- Arithmetic Operators
  - =, +, - , *, / , %, ++, --
- Assignment Operators
  - =, +=, -=, *=, /=, %=
- Comparison Operators
  - <=, >=, ==, !=, === , <, >
- Logical Operators
  - &&, ||, !

Persistent

Let us discuss few operators - Arithmetic

- Addition (+) operator
  - concatenates values when used with strings (result will be a String type)
  - adds if all operands are numbers (result will be number data type)

"Java" + "Script" = "JavaScript“ //Both Strings

34+ “56" = “3456” //One is string

34+ 56 = 90 //Both are numbers

Comparison operators

- Strick Equality operator ‘===‘
  - first checks data types of operands
  - returns false if types are incompatible or different
  - compare values if types are matching or compatible

```
alert(100 === 100); // ??

alert(100 === “100”);  // ??
```

In the first case, output is true whereas false for the second case

Persistent

Comparison operators continued ..

- Equality operator '==' follows below rules if operands are of different data types
  - Number & String
    - String is converted into Number
  - Number & Boolean
    - Boolean is converted into Number
  - Boolean & String
    - Both Boolean and String will get converted into Numbers

```
alert(1 == 1); // ??

alert(1 == "1");  // ??

alert(1 == true);  // ??

alert("1" == true);  // ??
```

In all cases, output will be true

typeof operator

- Used to determine data type of a variable

```
var course = “Java”;
console.log(typeof course); // string

var points = 5;
console.log(typeof points); // number

var courses = [];
console.log(typeof courses); // object

var course;
console.log(typeof course); // undefined
```

Control statements : if-else

- if- else construct

The condition must result in Boolean **true/false**

```
var book = "maths";

if( book == "history" )
        document.write("<b>History Book</b>");

else if( book == "maths")
        document.write("<b>Maths Book</b>");

else
        document.write("<b>Unknown </b>");
```


Persistent

Control statements continued : switch-case

```
var grade='A';

document.write("Entering switch block");

switch (grade) {

    case 'A':           document.write("Good job");
                 break;
    case 'B':           document.write("Pretty good");
                 break;
    case 'D':           document.write("Not so good");
                 break;
    case 'F':           document.write("Failed");
                 break;
    default:            document.write("Unknown")

 }
```

This block will be executed when no matching case is found

Looping statements – for & while

- for loop
- while loop

```
for ( var num = 1;num<=10 ;num++ )
{
        document.write (num);
}


var num = 1;

while ( num <= 10 ) {

        document.write( num );

        num ++;
}
```

Looping statements continued : do-while

- do-while loop

```
num = 12;

do
{
        document.write ( num );

        num++;
}

while ( num <= 10 )
```

Looping statements continued : for – in

- Iterates over the enumerable properties of objects in an arbitrary order

- Object ‘obj’
  - whose enumerable properties are iterated
  - ‘prop’ refers to the name of the property
  - ‘obj[prop]’ is the value of the property ‘prop’ for object ‘obj’

```
for(var prop in obj) {

        // statements
        console.log(obj[prop]);
       //statements

}
```

Break and Continue

- break

  Terminates the current iteration and exits the loop

- continue

  Skips current iteration and executes with next loop index.

.

```
for (i=0; i<10; i++)
{
        if (i==3){

            break;

            x=x +i ;
}


for( x = 0 ; x <= 50 ; x++ )
 {
        if ( x % 5!= 0 )

            continue;

        else
            //do something
 }
```

Arrays in JavaScript

- A data structure which is :-
  - Ordered, based on insertion
  - Growable
  - Can even store elements of different data types
  - Data type is ‘Object’

- Uses indexes to store elements
  - starts from ‘0’ index

How to create Arrays ?

- Two approaches :-
  - Using ‘[]’ brackets
  - Array in built constructor
- Initial size is optional
  - if not given, empty arrays will be created
  - even if given, arrays can still grow

Size is not specified.

Can specify the size.

Create and Initialize

```
var courses = []; // empty array

var courses = new Array(); // empty array

var courses = new Array(5);

var courses = new Array(“Java”, ”C”);
```


Persistent

How to add elements to an array ?

- Using index-based approach
- Using push method
- Which one to use and why ??

```
courses[0] = “Java" ;

courses[1] = “C" ;

courses.push(“Java“) ;
```


Persistent

Arrays – Retrieval & Traversal

- Using index
- Any loop statements
  - for in loop, neater & clean approach
- Methods of in-built Array constructor
  - pop

```
console.log(courses[0]) // Java

console.log(courses[1]) // C

for(var e in courses)
        console.log(courses[e]);

console.log(courses.pop()) // C
```

Array – Commonly used methods

Method	Description
push()	Adds new elements to the end of an array, and returns the new length
pop()	Removes the last element of an array, and returns that element
concat()	Joins two or more arrays, and returns a copy of the joined arrays
indexOf()	Search the array for an element and returns its position
reverse()	Reverses the order of the elements in an array

Overview of Array API

Method name	Description
a.toString()	Returns a string with the toString() of each element separated by commas.
a.toLocaleString()	Returns a string with the toLocaleString() of each element separated by commas.
a.concat(item[, itemN])	Returns a new array with the items added on to it.
a.join(sep)	Converts the array to a string - values delimited by the sep param
a.pop()	Removes and returns the last item.
a.push(item[, itemN])	Push adds one or more items to the end.
a.reverse()	Reverse the array.
a.shift()	Removes and returns the first item.
a.slice(start, end)	Returns a sub-array.
a.sort([cmpfn])	Takes an optional comparison function.
a.splice(start, delcount[, itemN])	Lets you modify an array by deleting a section and replacing it with more items.
a.unshift([item])	Prepends items to the start of the array.

Strings

- Any text inside
  - single quote ‘’
  - double quote “”

- Useful properties and methods
  - toUpperCase()
  - toLowerCase()
  - charAt(position)
  - indexOf(searchString, startPosition)
  - concat()
  - trim()
  - toString()

```
var course = ‘Java’;
        or
var course = “Java”;

course.toUpperCase() // JAVA

course.toLowerCase() // java

course.charAt(0) // J

course.indexOf(‘a’,2) // 3

course.concat(‘C’) // JAVAC

course.trim() // JAVA

course.toString() // JAVA
```

Strings continued..

- Some more methods :-
  - valueOf()
  - match(regExp)
  - replace(regExp,replaceValue)
  - search()
  - split(separator, limit)
  - substring(startPosition, endPosition)
  - length

```
course.valueOf() // Java

course.match(/v/) // v

course.replace(/Java/,'JavaScript') // JavaScript

course.search(/v/) // 2

course.split('a') // J, v,

course.substring(1,3) // av

course.length // 4
```

eval()

- evaluates JavaScript code represented as a string
  - if the string represents an expression, eval() evaluates the expression
  - if the string represents one or more JavaScript statements, eval() evaluates all
  - the statements

eval(string)

- as a best practice, don’t use it because it executes the code with the privileges of the caller :-
  - if string is affected by a malicious party, it may end up running
  - malicious code on the user's machine with the permissions of
  - webpage / extension

Template Literals

- Template literals are indicated by enclosing strings in backtick characters and provide syntactic sugar for constructing single and multi-line strings.
- These literals may include interpolated expressions inserted via ${…}.
- Backslash is used to escape inside template literals.

```
const fName= ‘Sachin’;
console.log(`Hello  ${fName}!  How are you today?`);
        // Output
// Hello Sachin!  How are you  today?
```


Persistent

String Interpolation

- String Interpolation is a way to wrap any code within ${ …. } ,execute that code, and added to the string in place.

```
let firstName= “ECMA”;
let  lastName=“Script”;
console.log(`Hi! ${firstName} ${lastName} !`s);

// Hi ECMAScript
```

Custom Interpolation

- Customized and flexible interpolation is possible for any arbitrary method.

```
var  userName= {name: “Sachin”};
console.log(‘Thanks for Shopping,
${userName.toUpperCase()}.’);

// output
//”Thanks for Shopping, Sachin”
```


Persistent

Multiline Strings

- To deal with multiline strings in JavaScript required hacky workaround.
- Using a \(backslash) before each newline is the current solution to have multiline string in JavaScript.
- In ES6, having multiple strings is significantly simple. Simply include a newline where they are needed.
- All whitespace inside the backticks is part of the string, so be careful with indentation.

```
console.log(`text in line 1
            text in line 2`);

//output
    //text in line 1
    //text in line 2
```

Summary : Session

With this we have come to an end of our session, where we discussed :

- Evolution, history and characteristics of JavaScript
- Including JavaScript in web pages
- Various concepts like variables, data types, operators, statements, arrays, strings etc

At the end of this session, we expect you to :

- Understand all basic constructs of JavaScript
- Apply & implement these concepts as and when required

Appendix

- References
- Key Contacts

References and books Links

- https://msdn.microsoft.com/en-us/library/ie/6974wx4d(v=vs.94).aspx
- http://javascript.crockford.com/javascript.html
- https://developer.mozilla.org/en-US/Learn/Getting_started_with_the_web/JavaScript_basics

Reference Material : Books

Head First JavaScript Programming

- By: Eric T. Freeman; Elisabeth Robson
- Publisher: O'Reilly Media, Inc.

Professional: JavaScript® for Web Developers

- By: Nicholas C. Zakas
- Publisher: Wrox

Persistent

Thank you!

Persistent Interactive | Persistent University