JavaScript

The EasyLearn Academy

9662512857

Introduction to javascript…

- JavaScript is an interpreted scripting language, built on the ECMAScript standard.
- The full form of ECMA is **European Computer Manufacturer's Association**.
- ECMAScript is a Standard for scripting languages such as JavaScript.
- Javascript is a procedural language based on prototypes, imperative, weakly typed,and dynamic.
- It is client side scripting language and therefore its code execute on client side.
- Today, JavaScript can execute not only in the browser, but also on the server, or actually on any device that has a special program called the JavaScript engine.
- Javascript is supported by all major browsers and enabled by default.
- JavaScript Open and cross-platform scripting language.

What **(in browser/client side)** javascript can do?

- Add new HTML tag into the page, change the existing content/tag, modify styles. It is known as DOM (document object modling).
- Response to user actions for an example run some code when user clicks on button, capture pointer movements, handle key press event etc.
- Send requests over the network to remote servers, download and upload files (AJAX/ react JS).
- Get and set cookies, ask questions to the visitor, show messages.
- Now javascript can store upto 5 MB data on the client-side using local storage.
- Can do input validation.

What CAN'T in-browser JavaScript do?

- JavaScript's abilities in the browser are limited for the sake of the user's safety.
- It is used to prevent an evil webpage from accessing private information or harming the user's data.
  1. JavaScript on a webpage cannot read/write files on the hard disk, can not copy them or can not execute programs. It has no direct access to OS functions.
  2. Modern browsers allow it to work with files, but the access is limited and only provided if the user does certain actions, like "dropping" a file into a browser window or selecting it via an <input> tag.
  3. camera/microphone and other devices can be accessed after user's permission.

JavaScript Syntax:

- A JavaScript code is written between the <script>... </script> of HTML tags in a web page.
- **You can place the <script> tag containing your JavaScript anywhere within you web page**
- it should be kept just after the end of <body> tags.
- The <script> tag is used to inform browser that the code between is JavaScript code.
- It has 2 optional attributes.
- **language:** This attribute specifies what scripting language you are using. It should be **javascript**
- **type:** This attribute is what is now recommended to indicate the scripting language in use and its value should be set to "***text/javascript**.*
- Each line in javascript **should ended with;(semicolon).**
- **However it is not rule. You can skip it if you always put online one statement per line.**

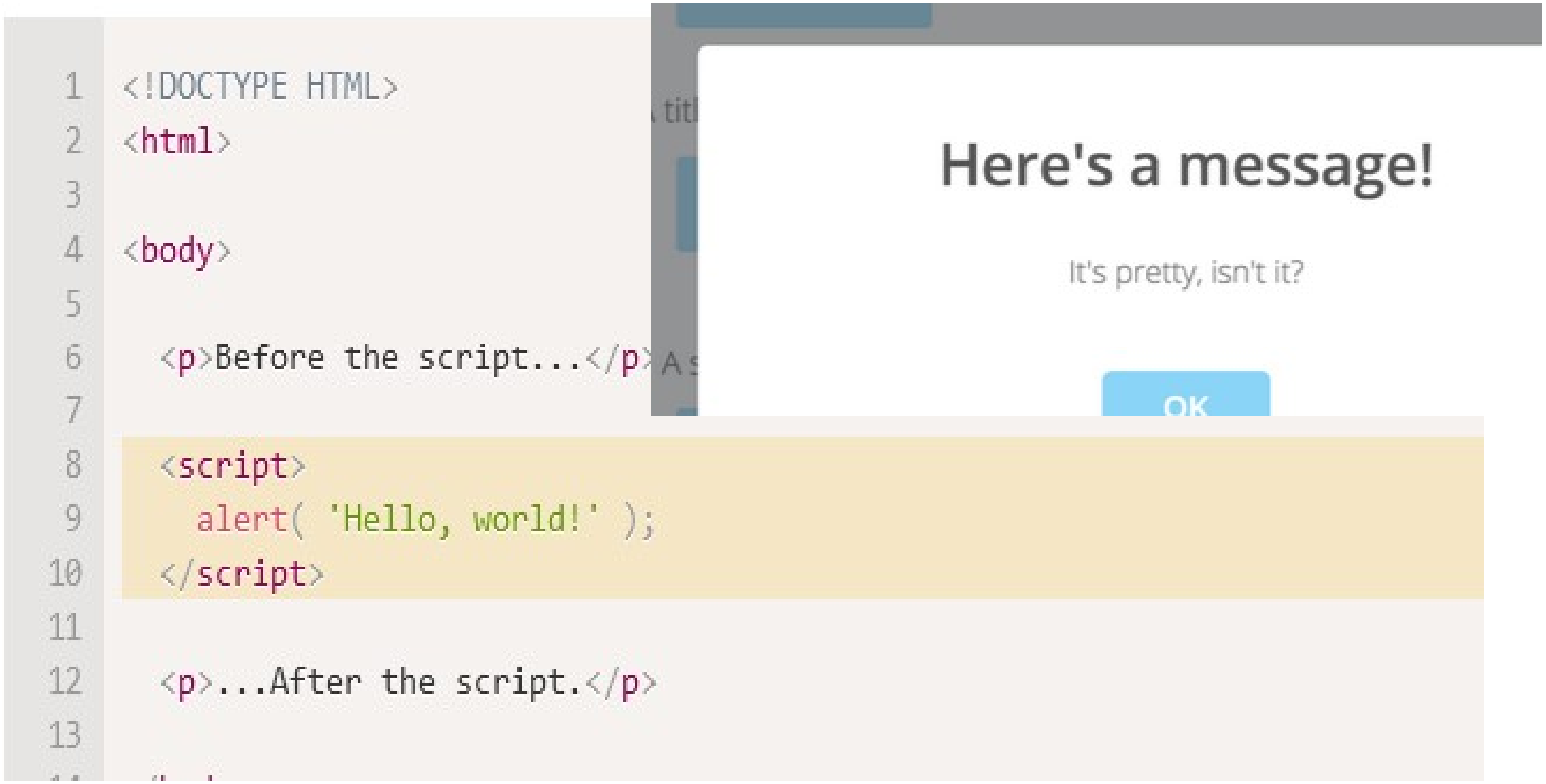
More about JavaScript

- JavaScript is case sensitive.
- This means that language **keywords**, **variables**, **function names**, and any other **identifiers** must always be typed with a consistent capitalization of letters.
- JavaScript supports both C-style and C++-style comments.
- Reusable JavaScript should be stored in separate file with .js extension.

35 Keywords in JavaScript

await	**break**	**case**	catch	class
continue	debugger	default	delete	**do**
export	extends	false	finally	**for**
function	**if**	import	in	instanceof
new	**null**	**return**	super	**switch**
this	throw	true	try	typeof
var	**void**	**while**	with	yield

Example of hello world



```
<!DOCTYPE HTML>
<html>

<body>

  <p>Before the script...</p>

  <script>
    alert( 'Hello, world!' );
  </script>

  <p>...After the script.</p>

```

Here's a message!

It's pretty, isn't it?

Alert Dialog Box

- An alert() is built in method used to give a small window along with some message to the users.
- It is used to show input error message when user give incorrect value in concern input control like textbox, textarea etc.

JavaScript Variables

- A variable is a “named storage” for data in main memory.
- Once we close web page/ browser, variables are deleted from memory. variables declared in one web page can not access from any other page.
- **Variable are data-type less.** It means we can store any type of value within it.
- Before we use a variable ,we **should** declare it.
- Variables are declared with the either **var** keyword or using **let** using **const** keyword. Initially we will use var keyword.
- For example

```
var name = “The EasyLearn Academy”
let age = 35;
const gender = true; //read-only variables
```

- There are two limitations on variable names in JavaScript:
  1. The name must contain only letters, digits, or the symbols **$** and **_**
  2. The first letter must not be a digit.

JavaScript DataTypes:

- There are 7 data types:
  1. **number** for both floating-point and integer numbers,
  2. **bigint** for integer numbers of arbitrary length,
  3. **string** for strings,
  4. **boolean** for logical values: true/false,
  5. **null** – a type with a single value null, meaning “empty” or “does not exist”,
  6. **undefined** – a type with a single value undefined, meaning “not assigned”,
  7. **object** – for complex data structures.

The "use strict" Directive

- "use strict"; Mean JavaScript code should be executed in "strict mode".
- It is not a statement, but a literal expression, ignored by earlier versions of JavaScript.
- in normal JavaScript, mistyping a variable name creates a new global variable.
- In strict mode, this will throw an error, making it impossible to accidentally create a global variable.
- So you can not use undeclared variables.
- *Use of undeclared variable are possible without use strict.*
- It should be given in double quote so IE 9 will not throw error.

```
"use strict";
pi = 3.14;       // it is an error because pi is not declared
alert(pi);
```

- If used inside a function, it has local scope it means only the code inside the function is in strict mode.
- Now let us see what is not allowed in strict mode.

What is not possible in strict mode.

- Using an object, without declaring it, is not allowed:

```
 match = {run:200,over:20};       // This will cause an error
```

- Deleting a variable (or object) is not allowed.

```
let pi = 3.14;
delete pi;
```

- Deleting a function is not allowed.

```
function add(x,y) {};
delete add;
```

- Duplicating a parameter name is not allowed:

```
function add(x,x) {};
```

- Octal numeric literals are not allowed:

```
let age = 010;             // This will cause an error
```

- Writing to a read-only property is not allowed:
- Writing to a get-only property is not allowed:
- The word eval cannot be used as a variable:
- The word arguments cannot be used as a variable:
- For security reasons, eval() is not allowed to create variables in the scope from which it was called:
- The this keyword in functions behaves differently in strict mode. The this keyword refers to the object that called the function.
- Keywords reserved for future JavaScript versions can NOT be used as variable names in strict mode. such as implements
- interface,let,package,private,protected,public,static,yield

Decision making statement

- JavaScript support following decision making statement.
  - If statement
  - If else statement
  - If elseif else statement
  - Switch statement
  - Nested decision making
- It syntax and nature is same as c/c++/java programming language.

loops

- In javascript we can use
  - While loop
  - For loop
  - Do while loop
- Syntax and cocept of above loops are same as c/c++/java.
- As well as there one more type of loop in javascript know as for each loop used only with objects.

Confirmation Dialog Box

- A confirmation dialog box is mostly used to take user's consent on any option. It displays a dialog box with two buttons: **OK** and **Cancel**.
- Example

```
<script>
var response = confirm("Do you want to continue ?");
if(response==true)
{
   alert("User wants to continue!");
}
else
{
   alert("User does not want to continue!");
}
</script>
```

Prompt Dialog Box

- Prompt is built-in function used to accept input from user using keyboard.
- It has 2 argument, 1st argument is message to be displayed and 2nd argument is default value which will be return in case user do not give input.
- It has following syntax.

```
<script>
var message = "Enter your name :”
var defaultvalue = "your name here"
var response = prompt(message,defaultvalue);
alert("You have entered : " + response);
</script>
```

How to display another page using javaScript? (without link/submit)?

- To display another page using javascript, we use following syntax
  - window.location = “relative page address";
- Example

```
<script type = "text/javascript">
     function Redirect() {
        window.location = "https://www.theeasylearnacademy.com";
     }
</script>
<form>
     <input type = "button" value = “open the easylearn academy site" onclick
     = "Redirect();" />
</form>
```

Document.write() Method

- The **write() method** is used add output at end of current html document.
- **Syntax:**
- document.write(variable1,expression,String, ...) **Parameters:** This method contain many parameters which is optional.
- All the expression arguments (exp1, exp2, …) can be listed and display with the order of occurrence.

Operators….

- JavaScript support following types of operators
  - Arithmetic Operators
    - Binary (+ - X / %)
    - Unary (++ --)
  - Comparison/Relational Operators
    - == != < > <= >= **===(exact equal to) !== (exact not equal to)**
  - Logical Operators
    - && (and) || (OR) ! (NOT)
  - Assignment Operators
    - = += -= *= /= %=
  - Conditional (or ternary) Operators
    - ? : (Conditional)

useful methods

- parseInt(value)
  - It is used to convert string into integer
- parseFloat(value)
  - It is used to convert string into float
- variable-name.toFixed(precision)
  - Round off float value upto given precision and return it as string

Some Strange Comparisons

- We all know how to compare values but let us understand some strange comparisons.
- We know that capital letter "A" is not equal to the lowercase "a". But Which one is greater? The lowercase "a‟ because the lowercase character has a greater index in the internal encoding table as JavaScript uses (Unicode).
- **When comparing values of different types, JavaScript converts the values to numbers.**

```
alert( '2’ > 1); // true, string '2' becomes a number 2
alert( '01' == 1 ); // true, string '01' becomes a number 1
```

- For boolean values, true becomes 1 and false becomes 0.

```
alert( true == 1 ); // true
alert( false == 0 ); // true
```

- A regular equality check == has a problem. It cannot differentiate 0 from false:

Some Strange Comparisons

```
alert( 0 == false ); // true
```

- The same thing happens with an empty string:

```
alert( '0' == false ); // true
```

- **This happens because operands of different types are converted to numbers by the equality operator ==. An empty string, just like false, becomes a zero.**
- What to do if we'd like to differentiate 0 from false?
- A strict equality operator === checks the equality without type conversion.

```
alert( 0 === false ); // false, because the types are different
```

- There is also a "strict non-equality" operator !== analogous to !=.
- There's a strange behavior when null or undefined are compared to other values.
- alert(null === undefined); // false
- For a non-strict check ==

```
alert( null == undefined ); // true
```

- There's a special rule.they equal each other
- For maths and other comparisons < > <= >=
- null/undefined are converted to numbers: null becomes 0, while undefined becomes NaN.
- Now let's see some funny things that happen when we apply these rules. And, what's more important, how to not fall into a trap with them.
- Let's compare null with a zero:

```
alert( null > 0 );  // (1) false
  alert( null === 0 ); // (2) false
  alert( null >= 0 ); // (3) true
```

- it is because The reason is that an equality check == and comparisons > < >= <= work differently.
- Comparisons convert null to a number, treating it as 0. That's why (3) null >= 0 is true and (1) null > 0 is false.
- On the other hand, the equality check == for undefined and null is defined such that, without any conversions, they equal each other and don't equal anything else. That's why (2) null == 0 is false.

Some Strange Comparisons

- The value undefined shouldn't be compared to other values:

```
alert( undefined > 0 ); // false (1)
alert( undefined < 0 ); // false (2)
alert( undefined == 0 ); // false (3)
```

- Why does it dislike zero so much? Always false!
- We get these results because:
- Comparisons (1) and (2) return false because undefined gets converted to NaN and NaN is a special numeric value which returns false for all comparisons.
- The equality check (3) returns false because undefined only equals null, undefined, and no other value.

- Why does it dislike zero so much? Always false!
- We get these results because:
- Comparisons (1) and (2) return false because undefined gets converted to NaN and NaN is a special numeric value which returns false for all comparisons.
- The equality check (3) returns false because undefined only equals null, undefined, and no other value.

How to avoid these problems

1. Treat any comparison with undefined/null except the strict equality === with exceptional care.
2. Don’t use comparisons >= > < <= with a variable which may be null/undefined, unless you’re really sure of what you’re doing. If a variable can have these values, check for them separately.

Summary

1. Comparison operators return a boolean value.
2. Strings are compared letter-by-letter in the “dictionary” order.
3. When values of different types are compared, they get converted to numbers (with the exclusion of a strict equality check).
4. The values null and undefined equal == each other and do not equal any other value.
5. Be careful when using comparisons like > or < with variables that can occasionally be null/undefined. Checking for null/undefined separately is a good idea.

Conditional operator '?'

```
var accessAllowed=null;
var age = prompt('How old are you?', '');
 age = parseInt(age);
if (age > 18)
  accessAllowed = true;
else
  accessAllowed = false;
alert(accessAllowed);
```

- The so-called "conditional" or "question mark" operator lets us do that in a shorter and simpler way.
- The operator is represented by a question mark ?. Sometimes it's called "ternary", because the operator has three operands.
- The syntax is:

```
var variable-name = (relational expression) ? value1 : value2
```

- example

```
let accessAllowed = (age > 18) ? true : false; //age 15
```

- can be also given like below

```
let accessAllowed = age > 18 ? true : false;
```

- it means parenthesis are optional
- A sequence of question mark operators ? can return a value that depends on more than one condition.
- A sequence of question mark operators ? can return a value that depends on more than one condition.

```
let age = parseInt(prompt('age?', 18));
let message = (age < 3) ? 'Hi, baby!' :
  (age < 18) ? 'Hi Teenager!' :
  (age < 100) ? 'Greetings!' :
  'What an unusual age!';
alert( message )
```

Nullish coalescing operator '??'

- the nullish coalescing operator is written as two question marks ??.
- As it treats null and undefined similarly, we'll use a special term here, in this article. We'll say that an expression is "defined" when it's neither null nor undefined.
- The result of a ?? b is:
-     if a is defined, then a,
-     if a isn't defined, then b.
- In other words, ?? returns the first argument if it's not null/undefined. Otherwise, the second one.
- it actually get the first "defined" value of the two.
- We can rewrite result = a ?? b using the operators that we already know, like this:

```
var user = ‘ankit’;
// if(user === null)
//     alert('Welcome unknown person');
// else
//     alert(user);
alert(user ?? 'Welcome unknown person');
```

What is document object?

- When an HTML document is loaded into a browser window, document object is created automatically.
- The ‘document’ object has various properties & methods that refer to other objects which allow access to and modification of the content of the web page.
- If there is a need to access any element in an HTML page, we always start with accessing the ‘document’ object.

Methods of Document Object:

1) **getElementById():** returns the element having the given id value.
2) **getElementsByName():** returns all the elements as **collection** whose name is given.
3) **getElementsByTagName():** returns all the elements as collection whose tag name is given.
4) **getElementsByClassName():** returns all the elements as collection whose class name is given.

What do we do once html tag is selected?

- We can get text, set text, change style, handle events etc, once html tag is selected.
- To get/set text from html tag, we either use innerHTML property if tag is output related tag or we use value properly if tag is input tag (input/textarea/select).
- To change style we style property. Means we can apply css using it's property name
- To change class name we use className property.
- To handle event we use event handler property name such as onclick.

innerHTML property

- For output related tags like div, p, span tr, td we use The **innerHTML** property.
- innerHTML property do not work with input, select and textarea tag because these tags are used to take input.
- To get text inside html tag innerHTML property is used on the right side of (=) operator.
- To set(change) text of html tag innerHTML property is used on left side of (=) operator.
- Example

```
<p id="content"></p> <!- non input tag like p,span,div tr, td b,I,c -->
<script>
    //change text of paragraph tag
     document.getElementById('content').innerHTML = "<b>learning javascript</b>";
     //get text of html tag
     var text = document.getElmentById('content').innerHTML;
     alert(text);
</script>
```

value property.

- We always use value property to get or set value of input related tag.
- Input, textarea and select, button tags are called input tags.
- To get text inside input tag valueproperty is used on the right side of (=) operator.
- To set(change) text of html tag value property is used on left side (=) operator.

```
<input type=‘text’ id=‘name’ value=‘the easylearn’ />
<script>
  var name = document.getElementById(‘name’).value;
  alert(name);
 document.getElementById(‘name’).value = ‘Ankit Patel’
</script>
```

Looping structure

- JavaScript support following loop statement
  - While loop (entry control loop)
  - For loop (entry control loop)
  - Do while loop (exit control loop)
  - For in loop (its not available in java)
- It syntax and nature is same as c/c++/java programming language.

External scripts

- **if we have a lot of JavaScript code, we can put it into a separate file.**
- Script files are attached to HTML with the src attribute:

```
<script src=“js/table-generator.js"></script>
```

- One can add more then once external javascript file.
- One can give reference of file from some other computer(server).
- It can be either placed in head section or just before completion of body tag.
- Best practices is to put before completion of body tag of page so page can be displayed to user at earliest.
- The benefit of a separate file is that the browser will download it and store it in its cache. so the file is actually downloaded only once.
- This decrease amount of data transferred (traffic) between client & server so the pages will display faster.

Functions….

- In **JavaScript**, a named section of a program that performs a specific task is called function.
- **function** is a one type of procedure or routine.
- Functions returns the value while procedure do not.
- Because of function/procedure we don’t have to write the same code again and again.
- **One Function should do only one thing.**
- Each function has name, some optional input, process and optionally return output
- It help is reduce code size.
- in JavaScript, you can create your own function to perform some specific task.
- Such function is called user defined function.
- To create a function we can use a function declaration.
- syntax

```
function function-name(parameter1, parameter2, ... parameterN)
{
    //function body .....
}
```

- example

```
function Greetings() {
  alert( 'Hello everyone!' );
}
Greetings(); //calling functions
```

- A variable declared inside a function is only visible inside that function.

```
function Greetings() {
  let message = "Hello, Everyone"; // local variable
  alert(message);
}
Greetings(); // calling function
alert( message ); // <-- Error! can't access message outside function
```

Another way to create functions

- There is one more way to create function. Known as function expression
- **A Function Declaration(1st method) can be called earlier than it is defined.**
- Function Expressions are created when the execution reaches them. So you can’t call it before it is created.

Syntax

```
let function-name =  function() {
// your code goes here….
}
```

- Example

```
let Greetings = function() {
  alert( "Hello Coder" );
};
Greetings(); //calling function
```

- We can copy a function into another variable (creating function alias):

```
let copy_function = Greetings;    // we create copy of greeting function as copy_function.
copy_function(); // now we can call greeting function using copy_function also.
greetings(); // calling function normally
```

Arrow functions

- There's another very simple and concise syntax for creating functions, that's often better than Function Expressions.
- It's called "arrow functions",
- syntax
- let function-name = (arg1, arg2, ..., argN) => one line expression
- example

```
let sum = (a, b) => a + b;
alert( sum(1, 2) ); // 3
example 2
let square = number => number * number;
alert(square(3) ); // value of n will be 9
```

Different between let and var

- Variable created using var has global scope.
- It means such variable can be accessed/changed from any functions as well as outside of functions

```
 var counter = 1; //global variable
//definition of function (created function)
function show_counter() {
    let step = 1; //local variable
    alert("from show counter function " + counter);
    counter = counter + step;
}
 show_counter();     //calling function
alert("outside function show counter " + counter);
console.log("step = " + step); //will cause error because step is local variable
```

- Variable declared using let is block scoped. Means **such variable can be accessed/changed only from block in which is created**.
- If you create variable using let keyword inside function, conditional statement, loop it can be accessed only inside that block. See example.

```
for (let i = 0; i < 5; i++) {
   console.log("Inside the loop:", i);
}
console.log("Outside the loop:", i); //will generate error
```

functions can access outer variable.

- A function can access an outer variable as well, for example:

```
let ClassName = 'The EasyLearn Academy'; //global variable
function Greetings() {
  let message = 'Hello, ' + ClassName; //accessing global variable
  alert(message); // Hello The EasyLearn Academy
}
Greetings();
```

- The function has full access to the outer variable. It can modify it also.
- Global variables are visible inside any function (unless same variable is declared using let keyword).
- It’s a good practice to minimize the use of global variables.
- Modern code has few or no global variables.
- Most variables declared in their functions are local.

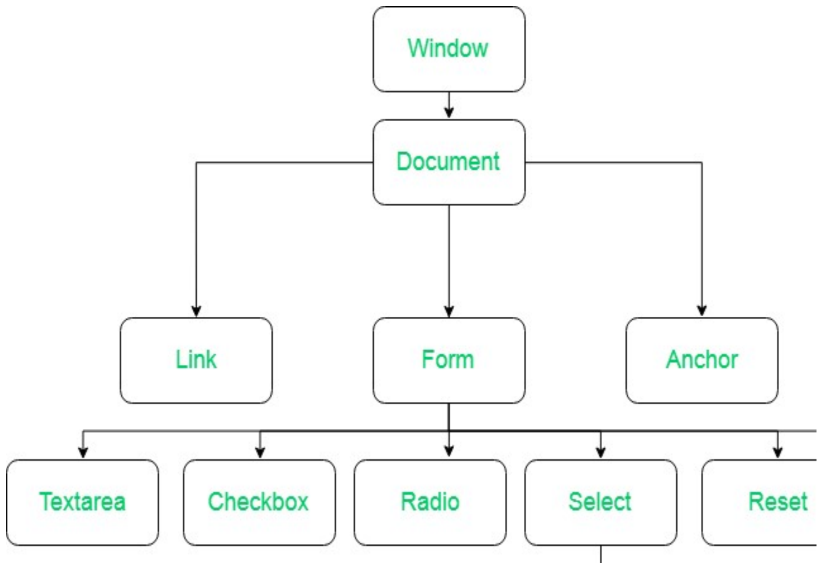
Default arguments in functions.

- Default arguments are used to provide value to arguments when function is called without passing argument
- If we don’t pass value for default argument then value given in function definition(default value) will be assigned to variable.

```
function power(base,exponent=2)
{
    return Math.pow(base,exponent)
}
alert(power(2,5)) #calling function with 2 arguments
alert(power(2)) #calling function with 1 arguments so value of exponent will be 2
```

DOM (document object modeling)

- an HTML document is made of tags. These tags are nested in each other. For example html tag has head and body tag.
- It means head and body are Nested tags /children of the body tag.
- **Each and every tag in webpage is object.** the **text inside a tag** is an **object** as well.
- All these objects can be **accessed/changed/removed** using JavaScript with the concept of **DOM**.
- The DOM represents HTML as a tree structure of tags.
- Event listeners (functions) can be added to nodes (html tag) and triggered on an occurrence of a given event.



What is an Event ?

- JavaScript's interaction with HTML is handled through events that occur when the user or the browser make changes in a page.
- Some example of events are
  1. When the page loads, it is called an event.
  2. When the user clicks a button,
  3. Other examples include events like pressing any key, closing a window, resizing a window, etc.
- Developers can use these events to execute JavaScript code as a responses,
- Events are a part of the Document Object Model (DOM)
- every HTML element contains a set of events which can trigger JavaScript Code.
- There are lots of event in javascript.
- Let us see some very important events

Mouse related events

Event name	**Event Handler property**	**Event Description**
click	onclick	When mouse click on an element
mouseover	onmouseover	When the cursor of the mouse comes over the element
mouseout	onmouseout	When the cursor of the mouse leaves an element
mousedown	onmousedown	When the mouse button is pressed over the element
mouseup	onmouseup	When the mouse button is released over the element
mousemove	onmousemove	When the mouse movement takes place.

keyboard related events

Event Name	Event Handler property	Event Description
Keydown & Keyup	onkeydown & onkeyup	When the user press and then release the key

Form related events

Event Name	**Event Handler property**	**Event Description**
focus	onfocus	When the user focuses on an element
submit	onsubmit	When the user submits the form
blur	onblur	When the focus is away from a form element
change	onchange	When the user modifies or changes the value of a form element

Window/Document events

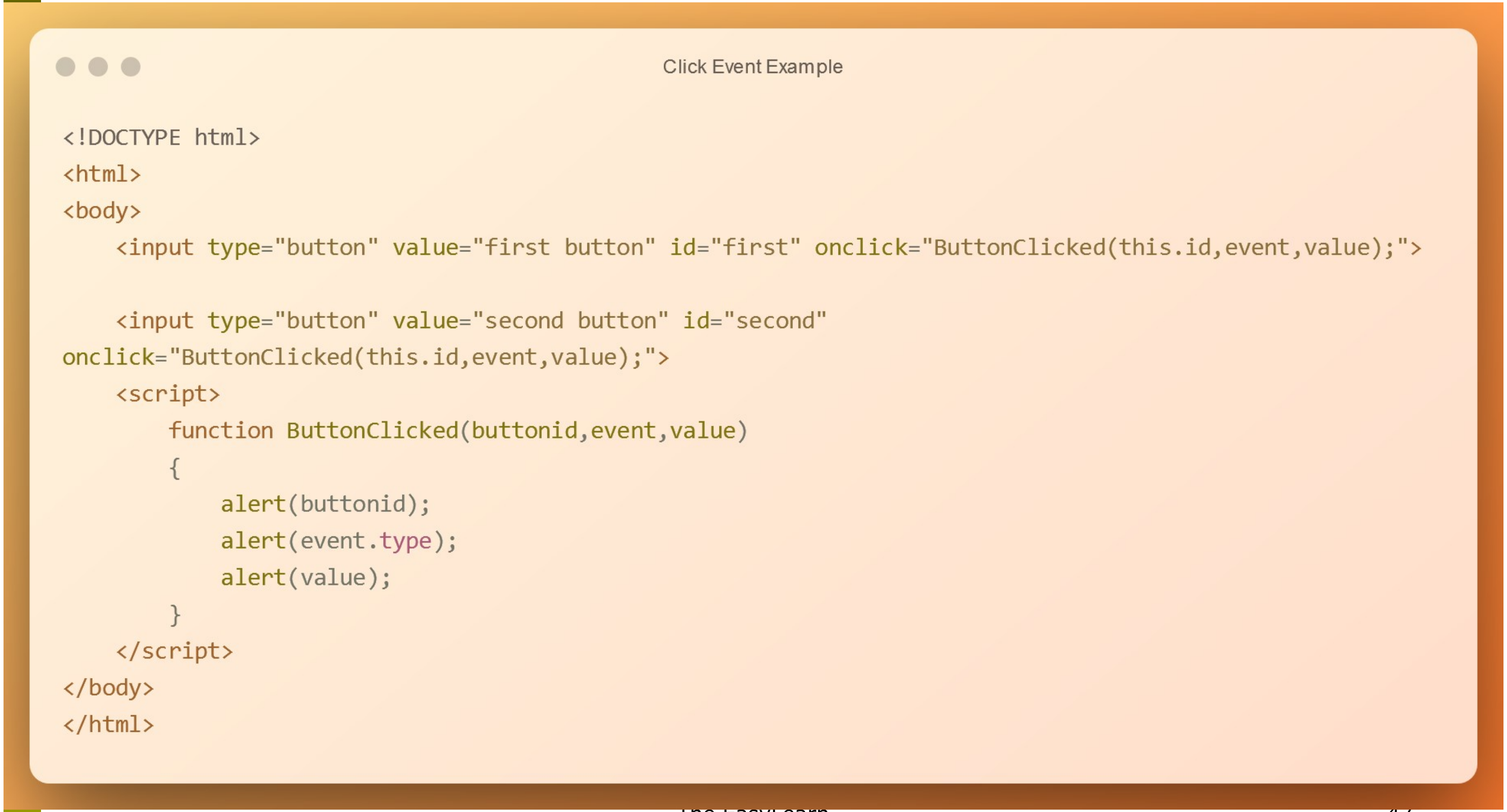
Event Name	Event Handler property	Event Description
load	onload	When the browser finishes the loading of the page
unload	onunload	When the visitor leaves the current webpage, the browser unloads it
resize	onresize	When the visitor resizes the window of the browser
Scroll	onscroll	When user scroll the web page.

There are three ways to assign event handlers.

1) HTML event handler attributes

- In this method we use **html event handler property inside html tag** to handle specific event.
- For example to handle button click event, we use onclick property. We give function name inside onclick property which will execute when user click on button.
- In this function we can optionally pass button id,event object and button’s value.
- This technique should be not used because of following issues.
  - the event handler code is mixed with the HTML code, which will make code difficult to maintain and extend.
  - timing issue. If the element is loaded fully before the JavaScript code, users can start interacting with the element on the webpage which will cause an error.
- Let us see some examples

onclick Event

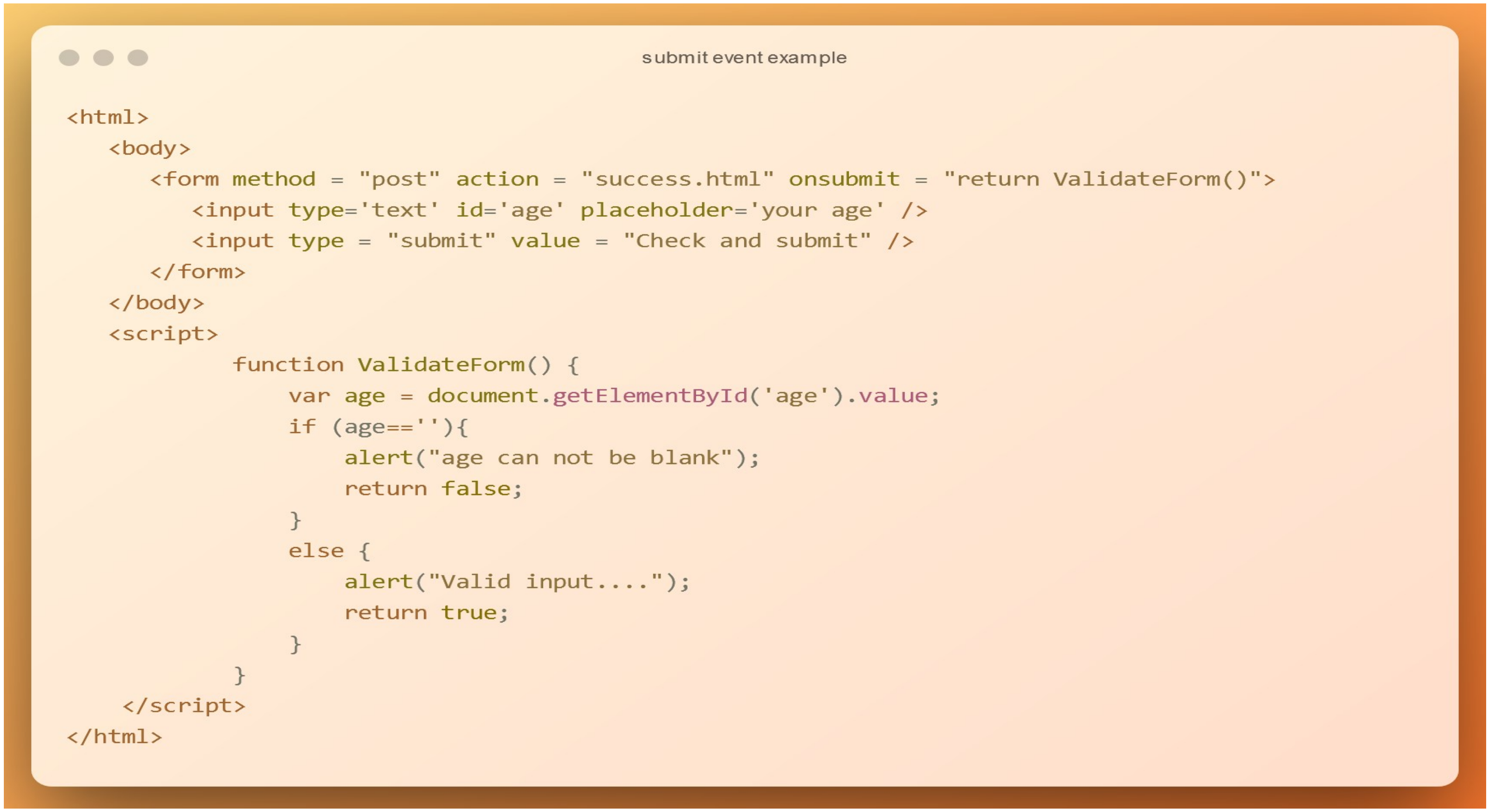

Click Event Example

```
<!DOCTYPE html>
<html>
<body>
    <input type="button" value="first button" id="first" onclick="ButtonClicked(this.id,event,value);">

    <input type="button" value="second button" id="second"
onclick="ButtonClicked(this.id,event,value);">
    <script>
        function ButtonClicked(buttonid,event,value)
        {
            alert(buttonid);
            alert(event.type);
            alert(value);
        }
    </script>
</body>
</html>
```

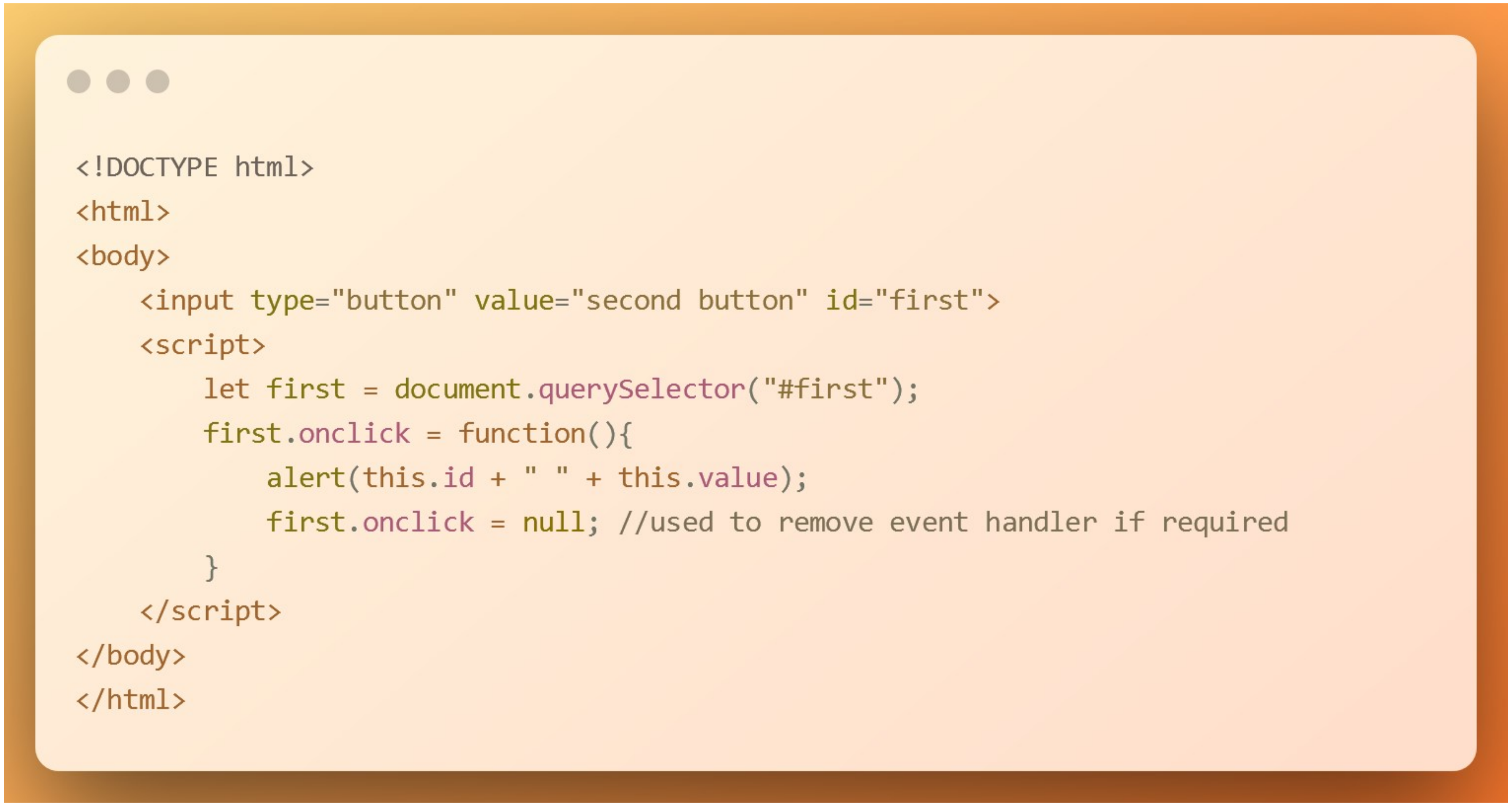
onsubmit Event

- **onsubmit** is an event trigger when you try to submit a form using submit button.
- You can make form validation using onsubmit event.



```
<html>
   <body>
      <form method = "post" action = "success.html" onsubmit = "return ValidateForm()">
         <input type='text' id='age' placeholder='your age' />
         <input type = "submit" value = "Check and submit" />
      </form>
   </body>
   <script>
            function ValidateForm() {
                var age = document.getElementById('age').value;
                if (age==''){
                    alert("age can not be blank");
                    return false;
                }
                else {
                    alert("Valid input....");
                    return true;
                }
            }
    </script>
</html>
```

2nd way of handling event (DOM Level 0 event handlers)



```
<!DOCTYPE html>
<html>
<body>
    <input type="button" value="second button" id="first">
    <script>
        let first = document.querySelector("#first");
        first.onclick = function(){
            alert(this.id + " " + this.value);
            first.onclick = null; //used to remove event handler if required
        }
    </script>
</body>
</html>
```

3rd way to handle event (DOM Level 2 event handlers)

- DOM Level 2 Event Handlers provide two main methods for dealing with the registering/deregistering event listeners:
  - addEventListener() – register an event handler
    - The addEventListener() method accepts three arguments: an event name, an event handler function, and a Boolean value that instructs the method to call the event handler during the **capture phase** (true) or during the **bubble phase(default) (false)** .
    - In bubble phase of html object are overlapping and if event is triggered for top most element then top most element event will execute first and bottom most element event will be triggered last.
    - In case of capture phase it is reverse of bubble phase.
  - removeEventListener() – remove an event handler**.**
- This is most preferred way of handling event in javascript.
- Let us see example

```html
<html>
<head>
    <style>
        div {
            color: white;       display: flex;      justify-content: center;
            align-items: center;    flex-direction: column;
        }
        h2 {    color: black;   }
        #grandparent {
            background-color: orange;       width: 300px;       height: 300px;
        }
        #parent {
            color:black !important;     background-color: white;
            width: 200px;                   height: 200px;
        }
        #child {
            background-color: green;    width: 100px;   height: 100px;
        }
    </style>
</head>
<body>
    <div>
        <div id="grandparent">GrandParent
            <div id="parent">Parent
                <div id="child">Child</div>
            </div>
        </div>
    </div>
    <script>
        let grandParent = document.getElementById("grandparent");
        let parent = document.getElementById("parent");
        let child = document.getElementById("child");
        grandParent.addEventListener("click",function() {
            console.log("GrandParent");
        },true);
        parent.addEventListener("click", function() {
            console.log("Parent");
        },true);
        child.addEventListener("click", function() {
            console.log("Child");
        },true);
    </script>
</body>
</html>
```

What is debugging?

- Debugging is the process of finding and fixing errors within a code.
- All modern browsers support debugging tools
- in most of the browsers there is a special UI in developer tools that makes debugging much easier.
- It also allows to trace the code step by step to see what exactly is going on.
- let us see practically

Objects

- A JavaScript object is special type of variable which has variables and methods.
- Variables in object are also called properties or state while method in object is also called behavior.
- **JavaScript is template based not class based**. Here, we don't create class to get the object. so, we directly create objects.
- For example: we can have MyMath object which has following variables & methods.
  - Variable : num1, num2
  - Methods : getAddition(), getSubstraction(), getMultiplication(), getDivision()

How to create object (1st technique)

- This if first technique to create object known as Using object literal
- syntax

```
var objectname = {
  variable-name: value,
  variable-name: value,
  method-name: function(){
  //code
},
  method-name: function(){
  //code
}
}
```

- Here, an object object_name is name of object.
- Each member of an object is a key: value pair separated by commas and enclosed in curly braces {}.
- **we can not create multiple variable of object type using this technique.**
- Example

```
var course = {
//property variables
name: “JavaScript",
duration:90,
display:function(){
  console.log(this.name,this.duration);
}
};
```

- You can also define an object in a single line.

```
var course = { name: “JavaScript” , trainer : "The EasyLearn Academy“,duration:90};
```

2nd technique to create object

- It is known as Create an Object using Instance of Object Directly.
- In this technique also, we can create multiple variable of object type known as object variables.

```
var person = new Object ( {
    //instance variable/property
    name: 'Ankit',
    age: 35,
    display: function() {
        console.log(“name ” + this.name + “ age “ + this.age);
    }
});
//var object-variable-name = ObjectName
var p1 = person;
console.log(“name “ + p1.name + “ Age ” + p1[‘age’]);
p1.display();
```

3rd way to create object

- 3rd technique of creating object uses function with argument to create object.
- It is also known as create JavaScript object using instance of an object

```
function Book(nm,pr,au) {
    //proerty variables
    this.name = nm,
    this.price = pr,
    this.author = au,
    this.display = function() {
        console.log(this.name,this.price,this.author);
    },
}
var b1  = new Book(‘Javascript’,500,’Ankit Patel’);
console.log(b1.name,b1.price,b1.author);
b1.display();
```

ES6 way to create class (4th technique)

- When we need to create multiple object that has different states but same methods we need to create class and then we use class to create object. For example

```
class Car {
  constructor(name,color,type) {
    this.name = name;
    this.color = color;
    this.type = type
  }
  display() {
    console.log(this.name + " " + this.color + " " + this.type);
  }
}
//var variable-name = new classname()
var car1 = new Car("Ford","Black","Petrol");
car1.display();
var car2 = new Car("Maruti","Red","Diesel");
car2.display();
```

What is inheritance?

- Inheritance is process of creating new class from the existing class.
- Existing class is called base class, or parent class or super class.
- Newly created class (which inherit other existing class) is called derived class or child class or sub class.
- Concept of inheritance increase reusability of code. It means we don’t have to copy & paste or have to write same code multiple times.
- The real benefit of inheritance is we can directly call the methods of parent class from child class’s method without creating any object of parent class.
- Child class method can also access & change properties of the parent class.
- Properties means variables declared inside the class.

How to create inheritance? (syntax)

```
class parent-class-name
{
    //properties
  //constructor
  //methods
}
class child-class-name extends parent-class-name
{
    //properties
  //constructor
  //methods
}
```

Inheritance in javascript

- Like c++, java one can use concept of inheritance in javascript. Let us see example.

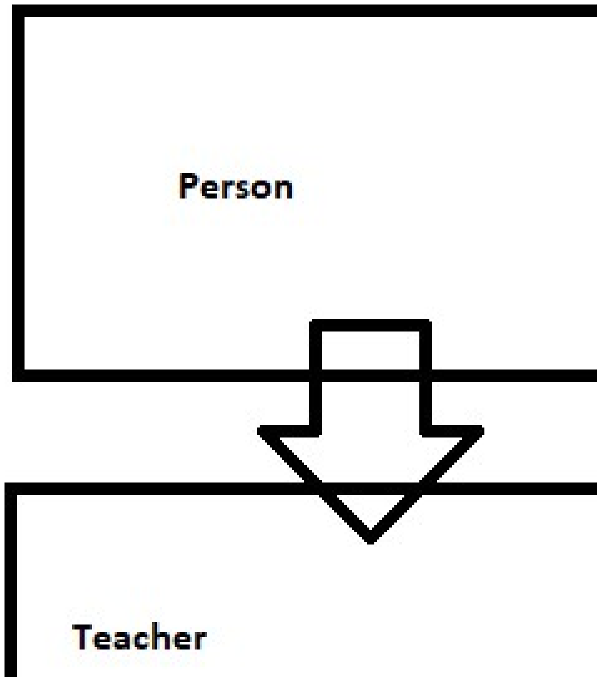
```
// parent class
class Person {
    constructor(name) {
        this.name = name; //property variable always start with this.
    }
    walk() {
        console.log(`${this.name} can walk`);
    }
}
// inheriting parent class
class Student extends Person {
    constructor(name,language)
    {
        super(name); //calling parent class constructor
        this.language = language; //property variable always start with this.
    }
    read()
     {
        console.log(`${this.name} read ${this.language}`);
    }
}
var s1= new Student('Ankit',’Gujarati’); //creating object
s1.walk(); //calling parent class method using child class object
s1.read();
```

Types of inheritance

- There are 5 types of inheritance.
  - Single level inheritance
  - Multilevel inheritance
  - Multiple inheritance
  - Hierarchical inheritance
  - Hybrid inheritance

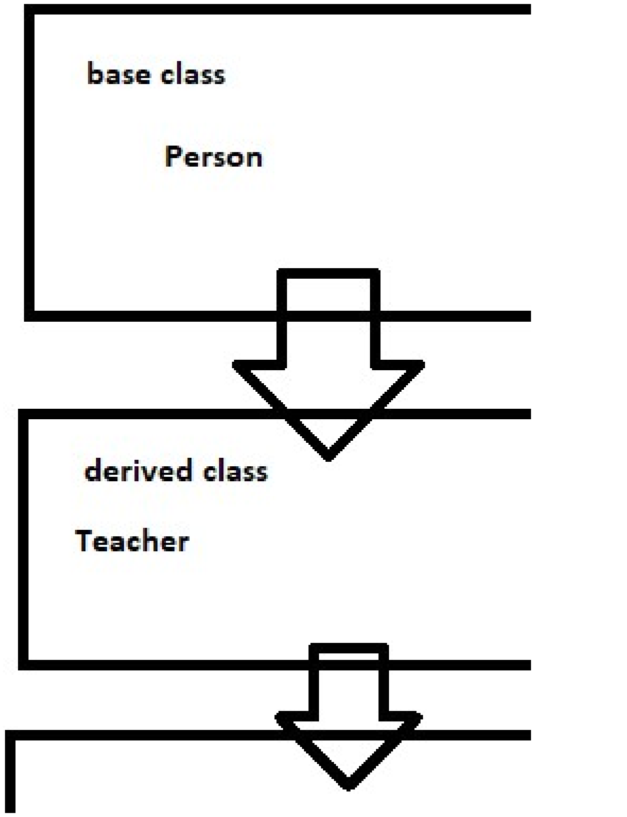
Single level inheritance

- When we create one new class from one existing class, it is called single level inheritance.
- There are always exactly two classes in single level inheritance.
- One is parent class while another is child class.



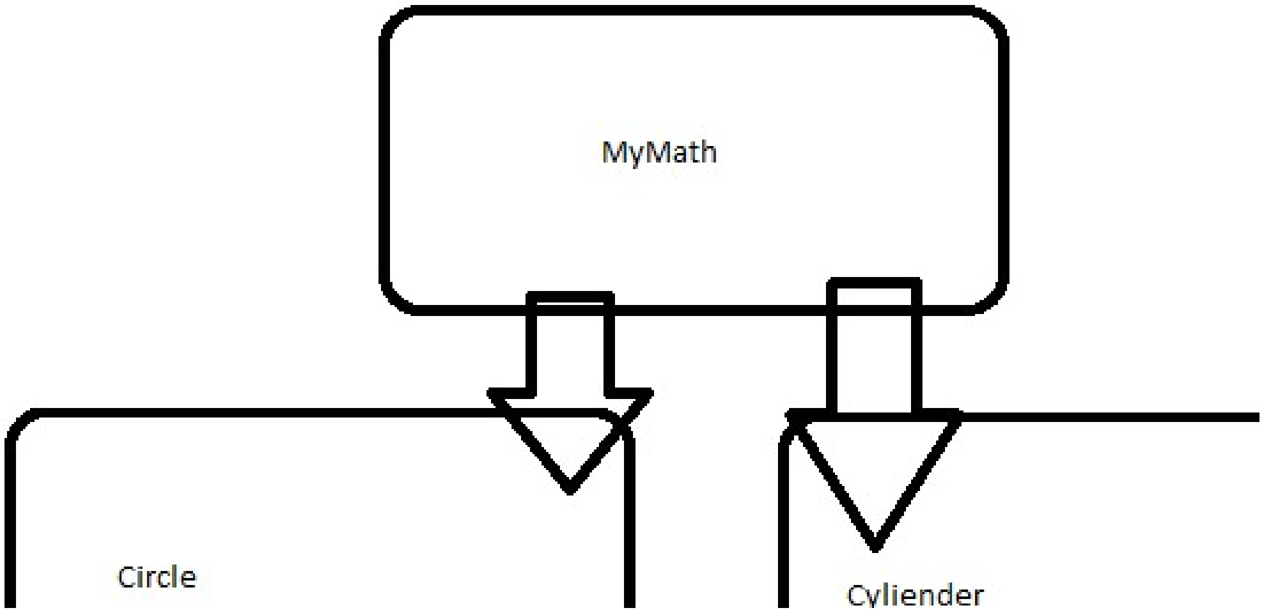
Multi-level inheritance

- When we create one new class from **derived class**, it is called Multilevel inheritance.
- For an example we have person class which is inherited(referenced) in teacher class which is inherited(referenced) in principal class, this creates Multi-level inheritance.
- There are always at least 3 level & 3 classes in Multi-level inheritance



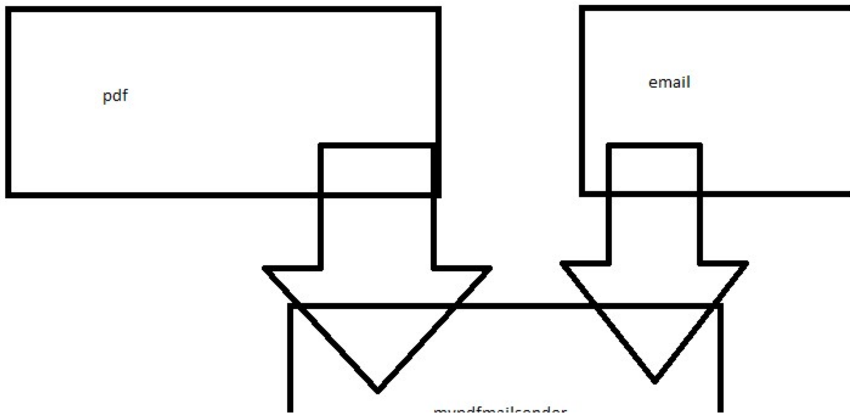
Hierarchical inheritance

- When we create more then one class from one existing class it is called Hierarchical inhertiance.
- There are always exactly one parent class and more then one child classes in Hierarchical inheritance.
- There are exactly two level in Hierarchical inheritance.



Multiple inheritance

- When we create one new class using two or more parent class (which are at same level), then it is called Multiple inheritance
- There are always 2 level in Multiple inheritance
- Parents are at level 1 while child is at level 2.



Hybrid inheritance

- When we use two different types of inheritance to create class hiearchy, it is called hybrid inheritance.
- There can be any number of levels in called hybrid inheritance.
- There is also not possible to give exact definition of hybrid inheritance

How to create empty object?

- An empty object (“empty cabinet”) can be created using one of two syntaxes:

```
let user = new Object();
// “object constructor” Syntax
let user = {};
// "object literal" syntax
```

How to add property into empty object?

- To add new property means instance variable into empty object,
- first we create empty object
  - let user = {}; //empty object
- Then we add properties into it.
  - user.name = “Ankit Patel”
- Or
  - user[‘age’] = 38;
- Using any one of the above method we can add any number of properties into object

How to check whether property exist or not?

- A notable feature of objects in JS is that it’s possible to access any property.
- There will be no error if the property doesn’t exist!
- Reading a non-existing property just returns undefined.
- So we can easily test whether the property exists:

```
let user = {}; //empty object
if( user.gender === undefined ) // true means "no such property"
    alert("no such property")
else
    alert(user.gender) // alert(user[‘gender’])
```

- The syntax is:

  "key" in object

- For instance:

```
let book = { author: "Ankit Patel", price: 380};
if(“author” in book) //true
    alert(book.author);
if(“name” in book) //false
    alert(book[‘name’]);
else
    alert(‘name key does not exist; );
```

Function binding …

- In javascript each function is object and therefore we can assign function reference to another variable and use newly created variable to call function.
- **we can not use this keyword to refer the instance variable of the object, if we call function using newly created variable. If we do so we will get error.**
- To use this to refer the instance variable we have to use concept of method binding.
- First let us see an example without binding.

```
var user = {
  name: "ankit",
  display:function(){
    console.log(this.name);
  }
}
user.display()
var show = user.display; //creating alias display for user.display method
show() //we cant use this when function is called using reference
```

- Now let us see how binding should be done.
- var reference-variable-name= object.function-name.bind(object);
- Let us see an example

Function binding

It has to be done like this

```
let user = {
    name: 'Ankit Patel',
    display: function() {
        console.log(this.name);
    }
};
//syntax
var alias = object.method.bind(object)
var show = user.display.bind(user);
show();
```

Array/List

- array is special variable in which we can store multiple values of any type. Later on we can add new values, change existing values as well as delete values from array.
- **Array is dynamic in javascript means we can add or remove any number of values in it.**
- Each value in array can be accessed/changed/removed using its unique position which is called index. Index is positive integer value.
- Arrays in JavaScript has special functions or methods that allow you to do different things with the values (elements).
- There are many ways to create array in javascript.
- For example

  var fruits = **[];** //empty array/list

Function	Description	Syntax
push()	Adds elements to the end of an array.	array.push(element1, ...)
pop()	Removes the last element from an array.	array.pop()
shift()	Removes the first element from an array.	array.shift()
unshift()	Adds elements to the beginning of an array.	array.unshift(element1, ...)
concat()	Merges two or more arrays.	array.concat(array2, ...)
slice()	Returns a shallow copy of a portion of an array.	array.slice(start, end)
splice()	Adds/removes elements from an array.	array.splice(start, deleteCount, item1, ...)
indexOf()	Returns the first index of an element, -1 if not found.	array.indexOf(searchElement, fromIndex)
includes()	Checks if an array contains an element. return true or false.	array.includes(valueToFind)
find()	Returns the first element that satisfies a condition. let arr = [1, 2, 3, 4]; let found = arr.find(x => x > 2); // returns 3	array.find(callbackFn)
findIndex()	Returns the index of the first element that satisfies a condition.	array.findIndex(callbackFn)
map()	map function pass each and every value to function and execute it. It return new array	array.map(callbackFn)
filter()	return a new array with elements that matches with condition.	array.filter(callbackFn)
sort()	The sort() function in JavaScript sorts the elements of an array in place and changes the original array. By default, it converts elements to strings and sorts them in ascending order, which can lead to unexpected results with numbers.	array.sort(compareFunction)
reverse()	Reverses the elements in place.	array.reverse()
join()	Joins all elements into a string.	array.join(separator)

object destructuring

- Object destructing is process of creating variables using the properties of the object and storing value of the property into these variables.
- Suppose you have a person object with two properties: firstName and lastName.

```
var person = {
firstName: 'Ankit', lastName: 'Patel‘, age:39, gender:true, weight:85
};
```

- when you want to assign properties of the person object to variables, you typically do it like this:

```
var firstName = person.firstName;
var lastName = person.lastName;
```

- ES6 introduces the object destructuring syntax that provides an alternative way to assign properties of an object to variables:

```
var { firstName,lastName,age,gender,weight } = person;
console.log(firstname,lastname,age,gender,weight);
```

ES6 Spread Operator …

- The JavaScript spread operator (...) allows us to quickly copy all or part of an existing array or object into another array or object.

```
var first = [1, 2, 3]; //array
var second = [4, 5, 6]; //array
var third = [...first, ...second];
console.log(third)
```

- We can use the spread operator with objects too:

Example of ES6 Spread Operator with objects …

```
var currentCar = {
    make:'Maruti',
    seat:4,
    engine:'petrol',
    color:'white',
    transmission:'manual'
};
var newCar = {
    make:'Audi',
    seat:6,
    engine:'EV',
    color:'black',
    price:5500000
};
console.log(currentCar,newCar);
var myCar = {...currentCar,...newCar};
console.log(myCar);
```

Example of ES6 Spread Operator about how to add object into array

```
let result = [];
console.log(result);
let student1 = { name: 'ram', html: 60, css: 70, javascript: 80 };

let student2 = { name: 'shyam', html: 75, css: 65, javascript: 85 };
let student3 = { name: 'geeta', html: 80, css: 75, javascript: 90 };
//add object at the end
result.push(student1);
//add object at the end using es6 operator
 result = [...result,student3];
```

for each loop

- To process all keys of an object, there exists a special form of the loop: for..in. This is a completely different thing from the for(;;) construct that we studied before.
- The syntax:

```
for (let key in object) {
  // executes the body for each key among object properties
}
```

```
let user = {
  name: "Ankit",
  age: 35,
  gender: true
};

for (let key in user) {
  alert( key );  // name, age, gender
  alert( user[key] ); // Ankit, 35, true
}
```

Object references and copying

- One of the fundamental differences of objects versus basic type is that objects are stored and copied “by reference”, whereas basic type: strings, numbers, booleans, etc – are always copied “as a whole value”.

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

- As a result we have two independent variables, each one has string "Hello!".
- A variable assigned to an object stores not the object itself, but its “address in memory” – in other words “a reference” to it.
- Let’s look at an example of such a variable:

```
let user = {
  name: "Ankit“, surname : “patel”
};
```

- The object is stored somewhere in memory & the user variable has a “reference” to it.
- When an object variable is copied, the reference is copied, but the object itself is not duplicated.
- For instance:
- `let admin = user;`
- Now we have two variables, each storing a reference to the same object:
- We can use either variable to access the object and modify its contents:

```
admin.name = 'Ankit Patel'; // changed by the "admin" reference
alert(user.name);
```

how to make duplicate (clone) an object?

- to create duplicate object means independent copy also called clone we need to use trick.
- to create a new object and replicate the structure of the existing one one has to iterate over its properties and copying them on by one.
- Like this:

```
let user = {
  name: "Ankit",
  age: 35
};

let clone = {}; // the new empty object
// clone = user; // it will not copy user into clone, it store reference of
   user into clone, means if you change user , clone will change and vice
   versa
// let's copy all user properties into it
for (let key in user) {
  clone[key] = user[key];
}
// now clone is a fully independent object with the same content
clone.name = "Jiya Patel"; // changed the data in it
alert( user.name ); // still Ankit in the original object
```

const

- const is a keyword used to declare a **constant variable**, meaning its value cannot be reassigned after it's initialized.
- A variable declared with const must be assigned a value when it’s declared, and this assignment cannot be changed.
- However, if the const variable holds an **object** or **array**, the contents of the object or array can still be modified.
- const is **block-scoped**, meaning it only exists within the block ({...}) it was declared in. It’s not accessible outside of that block
- Let us see example

Const example

```
<script>
        const name = "Ankit";
        const fullname = “Patel ” + name;
        console.log(name,fullname);
        //name = "Ankit Patel";  //  Error: Assignment to constant
   variable.
           const user = { name: "Param“,age:19 };
        user.name = "Param Patel";  //  Allowed: modifying property of
   the object.
        console.log(user.name);  // Outputs: Param Patel
</script>
```

What are Cookies?

- Cookies are speical type of variables, stored in small text files, on client computer.
- Normal variables will be destroyed whenever we close webpage or browser, it is also reset when we refresh web page.
- While cookies will be preserved even after the web page is closed/refresh.

Why we need cookies variables?

- after web server send a web page to a client, connection is lost, and server does not remember any detail about the client.
- Now this can be very big problem as every time server receive new request from client, server has no idea about client. Means server does not know whether client is guest or premium user.
- To solve this problem cookie are used.
- Cookies are usually created/updated/deleted/accessed by both by server & client but it is do remember it stored in client’s memory.
- it is data in form of key value pair just like object-variable.
- Cookies should be encrypted to prevent unauthorized access to it by anyone.

How to Create/ Change a Cookie with JavaScript?

- To create/change cookie use below syntax. If cookie is already exist then it will be updated.

```
document.cookie = “classname=The EasyLearn Academy";
```

- You can also add an expiry date (in UTC time).:

```
document.cookie = " classname=The EasyLearn Academy;
   expires=mon, 15 Dec 2050 12:00:00 UTC";
```

- With a path parameter, you can tell the browser what path the cookie belongs to. By default, the cookie belongs to the current page.

```
document.cookie = " classname=The EasyLearn Academy;
   expires=mon, 15 Dec 2050 12:00:00 UTC; path=/";
```

How to read cookie?

```
var cookies = document.cookie; //return
  key value pair array
console.log(cookies)
```

How to delete cookie?

- Deleting a cookie is very simple.
- You don't have to specify a cookie value when you delete a cookie.
- Just set the expires parameter to a passed date:

```
document.cookie = "classname=The EasyLearn
Academy; expires=mon, 15 Dec 2000 12:00:00
UTC";
```

- Creating/editing/deleting cookies can be boring task if done manually, so it is better to create & use functions for these task.

Use setCookie function to create,edit,delete particular cookie & use getCookie function to get specific cookies value

```
//setCookie function is used to set/change/delete cookies
function setCookie(name,value='',days=100) {
    var d = new Date();
    d.setTime(d.getTime() + (days*24*60*60*1000));
    var expires = "expires="+ d.toUTCString();
    document.cookie = name + "=" + value + ";" + expires + ";path=/";
}
//getCookie function is used to get value of specific cookie by name
function getCookie(name) {
    var name = name + "=";
    var decodedCookie = decodeURIComponent(document.cookie);
    var ca = decodedCookie.split(';');
    for(var i = 0; i <ca.length; i++) {
      var c = ca[i];
      while (c.charAt(0) == ' ') {
        c = c.substring(1);
      }
      if (c.indexOf(name) == 0) {
        return c.substring(name.length, c.length);
      }
    }
    return "";
  }
```

How to print web page in javascript?

- Many times you would like to place a button on your webpage to print the content of that web page via an actual printer.
- This is possible to do using the **print** function of **window** object.Example

```
<form>
     <input type = "button" value = “Print " onclick = “window.print()" />
</form>
```