🌐 Complete JavaScript BOM & Events Guide

I'll explain **Browser Object Model (BOM)**, **Timing Functions**, and **Event Handling** in detail.

Part 1: Window Object & Opening Windows

1. Opening New Windows

```
var myWin;

function openWindow(){
   myWin = open("https://www.google.com", "_blank",
"width=300px,height=300px")
}
```

W3Schools Reference: `window.open()` opens a new browser window.

Syntax:

javascript

```
window.open(URL, name, specs, replace)
```

Parameters:

- `URL` : Website to open
- `name` : Target (`_blank` , `_self` , `_parent` , `_top`)
- `specs` : Window features (width, height, etc.)
- `replace` : true/false (replace history)

Scenarios:

javascript

```
// Open blank window
var win1 = open("", "_blank", "width=200,height=200")

// Open Google in new window
var win2 = open("https://www.google.com", "_blank", "width=500,height=500")
```

```
// Open with multiple features
var win3 = open("page.html", "_blank", "width=400,height=300,left=100,top=50")

// Write to new window
var win = open()
win.document.write("Hello World")
```

---

2. Closing Windows

javascript

```
function closeWindow(){
    myWin.close()
}
```

Important: Only works on windows opened by JavaScript, not browser tabs.

---

3. Moving Windows

moveTo() - Absolute Position

javascript

```
function MoveToWindow(){
    myWin.moveTo(400, 400)
}
```

- Moves window to **specific coordinates** (x, y)
- (0, 0) is top-left corner of screen

moveBy() - Relative Position

javascript

```
function MoveByWindow(){
    myWin.moveBy(300, 300)
}
```

- Moves window **relative to current position**
- Positive values = move right/down
- Negative values = move left/up

Scenarios:

javascript

```
// Move to top-left corner
myWin.moveTo(0, 0)

// Move to center of screen
myWin.moveTo(screen.width/2, screen.height/2)

// Move 100px right and 50px down
myWin.moveBy(100, 50)

// Move 100px left
myWin.moveBy(-100, 0)
```

---

4. Resizing Windows

resizeTo() - Absolute Size

javascript

```
function resizeToWindow(){
    myWin.resizeTo(100, 100)
}
```

- Sets window to **exact dimensions**

resizeBy() - Relative Size

javascript

```
function resizeByWindow(){
    myWin.resizeBy(100, 100)
}
```

- Changes size **relative to current size**

Scenarios:

javascript

```
// Set specific size
myWin.resizeTo(800, 600)

// Make bigger by 100px
myWin.resizeBy(100, 100)

// Make smaller by 50px
myWin.resizeBy(-50, -50)

// Maximize (full screen)
myWin.resizeTo(screen.width, screen.height)
```

Part 2: Timing Functions

1. setTimeout() - Execute Once After Delay

W3Schools Reference: `setTimeout()` executes code after specified milliseconds.

Syntax:

javascript

```
setTimeout(function, milliseconds, param1, param2, ...)
```

Method 1: Named Function

javascript

```
function test(){
    console.log("set Time Out Callback func");
}

setTimeout(test, 3000) // Executes after 3 seconds
```

Method 2: Anonymous Function

javascript

```
setTimeout(function(){
    console.log("Executed after 3 seconds")
}, 3000)
```

Method 3: With Parameters

javascript

```
function sayHello(trackName, track){
    console.log("Welcome " + trackName + track);
}

setTimeout(sayHello, 2000, "PD", "track3")
// After 2 seconds: "Welcome PDtrack3"
```

Scenarios:

javascript

```
// Simple delay
setTimeout(function(){
    alert("Time's up!")
}, 5000)

// Multiple executions
setTimeout(function(){
    test()
    test2()
}, 3000)

// Delayed redirect
setTimeout(function(){
    window.location = "https://google.com"
}, 3000)
```

---

2. clearTimeout() - Cancel setTimeout

javascript

```
var timer = setTimeout(function(){
    alert("This won't show")
```

```
}, 5000)

// Cancel before it executes
clearTimeout(timer)
```

---

3. setInterval() - Execute Repeatedly

W3Schools Reference: `setInterval()` executes code repeatedly at specified intervals.

javascript

```
var myInterval = setInterval(test, 2000)
// Executes test() every 2 seconds
```

Example: Counter

javascript

```
var span = document.getElementById("result")
var counter = 0
var myInterval;

function start(){
   myInterval = setInterval(function(){
        span.innerHTML = ++counter
    }, 1000) // Updates every 1 second
}

function end(){
    clearInterval(myInterval) // Stops the counter
}
```

Scenarios:

javascript

```
// Clock
setInterval(function(){
    var now = new Date()
    document.getElementById("clock").innerHTML = now.toLocaleTimeString()
}, 1000)
```

```javascript
// Auto-refresh page every 10 seconds
setInterval(function(){
    location.reload()
}, 10000)
```

---

4. clearInterval() - Stop Interval

javascript

```javascript
var myInterval = setInterval(test, 2000)

// Stop after 5 seconds
setTimeout(function(){
    clearInterval(myInterval)
}, 5000)
```

Practical Example: Download Link Timer

javascript

```javascript
var p2 = document.getElementById("result")

function download(){
    p2.innerHTML = "Link will appear after 3 seconds"

    setTimeout(function(){
        p2.innerHTML = "<a href='https://google.com'>Google</a>"
    }, 3000)
}
```

Flow:

1. Click download button
2. Shows "Link will appear after 3 seconds"
3. After 3 seconds, link appears

Part 3: Location Object

javascript

```
console.log(location)
```

W3Schools Reference: `location` object contains information about current URL.

Properties:

javascript

```
console.log(location.href)     // Full URL
console.log(location.hostname)  // Domain name
console.log(location.pathname)  // Path after domain
console.log(location.protocol)  // http: or https:
console.log(location.port)      // Port number
```

Methods:

reload() - Refresh Page

javascript

```
location.reload() // Refresh current page

// Auto-refresh every 3 seconds
setInterval(function(){
    location.reload()
}, 3000)
```

assign() - Navigate to New URL

javascript

```
location.assign("https://google.com")
// Can go BACK using browser back button
```

replace() - Replace Current Page

javascript

```
location.replace("https://google.com")
// Cannot go BACK (removes from history)
```

Difference:

Method	Back Button Works?	History
`assign()`	✅ Yes	Adds to history
`replace()`	❌ No	Replaces history

Scenarios:

javascript

```
// Redirect after login
setTimeout(function(){
    location.assign("dashboard.html")
}, 2000)

// Logout (no going back)
location.replace("login.html")

// Get current page info
if(location.pathname === "/admin"){
    console.log("Admin page")
}
```

Part 4: Screen Object

javascript

```
console.log(screen.width)     // Total screen width
console.log(screen.height)    // Total screen height
console.log(window.innerWidth)  // Browser window width
console.log(window.innerHeight) // Browser window height
```

W3Schools Reference: `screen` object contains information about user's screen.

Scenarios:

javascript

```
// Check if mobile
if(screen.width < 768){
```

```
    console.log("Mobile device")
}

// Center window on screen
var left = (screen.width - 500) / 2
var top = (screen.height - 400) / 2
window.open("page.html", "_blank",
`width=500,height=400,left=${left},top=${top}`)

// Responsive behavior
if(window.innerWidth < 600){
    document.body.style.fontSize = "12px"
}
```

Part 5: History Object

javascript

```
console.log(history)
```

W3Schools Reference: `history` object contains browser's history.

Methods:

javascript

```
history.back()     // Go back one page (like back button)
history.forward()  // Go forward one page
history.go(-1)     // Go back 1 page
history.go(-2)     // Go back 2 pages
history.go(1)      // Go forward 1 page
```

Scenarios:

javascript

```
// Custom back button
document.getElementById("backBtn").onclick = function(){
    history.back()
}
```

```
// Go back 3 pages
history.go(-3)
```

Part 6: Navigator Object

javascript

```
console.log(navigator)
```

W3Schools Reference: `navigator` object contains information about the browser.

Properties:

javascript

```
console.log(navigator.userAgent)    // Browser info
console.log(navigator.language)     // Browser language
console.log(navigator.onLine)       // Internet connection status
console.log(navigator.platform)     // Operating system
```

Geolocation API

javascript

```
function success(position){
    console.log(position.coords.latitude)
    console.log(position.coords.longitude)
}

function error(){
    console.log("User denied location access")
}

navigator.geolocation.getCurrentPosition(success, error)
```

W3Schools Reference: Geolocation API gets user's geographical position.

Scenarios:

javascript

```javascript
// Get user location
navigator.geolocation.getCurrentPosition(function(pos){
    var lat = pos.coords.latitude
    var lon = pos.coords.longitude
    console.log(`Location: ${lat}, ${lon}`)

    // Show on map
    var mapURL = `https://maps.google.com/?q=${lat},${lon}`
    window.open(mapURL)
})

// Check if online
if(navigator.onLine){
    console.log("Connected to internet")
} else {
    alert("No internet connection")
}

// Detect browser
if(navigator.userAgent.includes("Chrome")){
    console.log("Chrome browser")
}
```

Part 7: Event Handling

Method 1: Inline Events (❌ Not Recommended)

html

```html
<button onclick="changeContent()">Click Me</button>
```

Problems:

- Mixes HTML and JavaScript
- Can only assign ONE function per event
- Hard to maintain

Method 2: onclick Property

javascript

```
var btn = document.getElementById("result")
var p = document.getElementById("p1")

btn.onclick = changeContent
btn.onclick = changeStyle  // ❌ Overwrites changeContent!
```

Problem: Only ONE function can be assigned

Solution: Wrap in function

javascript

```
btn.onclick = function(){
    changeContent()
    changeStyle()
}
```

Method 3: addEventListener() (✅ BEST METHOD)

W3Schools Reference: `addEventListener()` attaches event handler to element.

javascript

```
btn.addEventListener("click", changeContent)
btn.addEventListener("click", changeStyle)
// Both functions execute! ✅
```

Syntax:

javascript

```
element.addEventListener(event, function, useCapture)
```

Advantages:

- Multiple functions per event
- Easy to remove with `removeEventListener()`

- Better control over event flow

Common Events:

Mouse Events:

javascript

```
btn.addEventListener("click", function(){
    console.log("Clicked")
})

btn.addEventListener("dblclick", function(){
    console.log("Double clicked")
})

btn.addEventListener("mouseenter", function(){
    console.log("Mouse entered")
})

btn.addEventListener("mouseleave", function(){
    console.log("Mouse left")
})

btn.addEventListener("mouseover", function(){
    console.log("Mouse over")
})
```

Keyboard Events:

javascript

```
input.addEventListener("keydown", function(e){
    console.log("Key pressed:", e.key)
})

input.addEventListener("keyup", function(e){
    console.log("Key released:", e.key)
})

input.addEventListener("keypress", function(e){
```

```
    console.log("Key press")
})
```

Form Events:

javascript

```
input.addEventListener("focus", function(){
    input.style.border = "5px solid green"
})

input.addEventListener("blur", function(){
    input.style.border = "1px solid gray"
})

input.addEventListener("input", function(){
    console.log("Input value:", input.value)
})

input.addEventListener("change", function(){
    console.log("Input changed")
})
```

Example: Dynamic Border Color

javascript

```
var input = document.getElementById("input1")

function changeBorder(){
    input.style.border = "5px solid green"
}

function changeBorderWithInput(){
    if(input.value.length > 3){
        input.style.border = "5px solid yellow"
    } else {
        input.style.border = "5px solid red"
    }
}
```

```
input.addEventListener("focus", changeBorder)
input.addEventListener("input", changeBorderWithInput)
```

Flow:

1. User clicks input → border turns **green**
2. User types 1-3 characters → border turns **red**
3. User types 4+ characters → border turns **yellow**

---

removeEventListener()

javascript

```
btn.addEventListener("click", changeStyle)

// Remove after 3 seconds
setTimeout(function(){
    btn.removeEventListener("click", changeStyle)
}, 3000)
```

Important: Function must be named (not anonymous) to remove

javascript

```
// ❌ Cannot remove
btn.addEventListener("click", function(){
    console.log("Click")
})

// ✅ Can remove
function handleClick(){
    console.log("Click")
}
btn.addEventListener("click", handleClick)
btn.removeEventListener("click", handleClick)
```

Part 8: Event Object

javascript

```
function changeContent(e){
    console.log(e)  // Event object
    p.innerHTML = "Hello PD"
}

btn.addEventListener("click", changeContent)
```

W3Schools Reference: Event object contains information about the event.

Event Object Properties:

javascript

```
function handleEvent(e){
    console.log(e.type)         // "click", "keydown", etc.
    console.log(e.target)       // Element that triggered event
    console.log(e.currentTarget) // Element with event listener
    console.log(e.clientX)      // Mouse X coordinate
    console.log(e.clientY)      // Mouse Y coordinate
    console.log(e.key)          // Key pressed (keyboard events)
    console.log(e.code)         // Physical key code
}
```

Scenarios:

javascript

```
// Get clicked element
document.addEventListener("click", function(e){
    console.log("Clicked:", e.target.tagName)
})

// Get mouse position
document.addEventListener("mousemove", function(e){
    console.log(`X: ${e.clientX}, Y: ${e.clientY}`)
})

// Detect which key was pressed
document.addEventListener("keydown", function(e){
    if(e.key === "Enter"){
        console.log("Enter pressed")
    }
    if(e.key === "Escape"){
        console.log("Escape pressed")
```

```
    }
})
```

Part 9: Event Propagation (Bubbling & Capturing)

HTML Structure:

html

```
<div class="first" id="first">
  First
  <div class="second" id="second">
    Second
    <div class="another" id="another">
      another
      <div class="third" id="third">Third</div>
    </div>
  </div>
</div>
```

Event Bubbling (Default Behavior)

javascript

```
var first = document.getElementById("first")
var second = document.getElementById("second")
var third = document.getElementById("third")

first.addEventListener("click", function(e){
    console.log("first")
})

second.addEventListener("click", function(e){
    console.log("second")
})

third.addEventListener("click", function(e){
    console.log("third")
})
```

When you click on "Third":

```
Output:
third
second
first
```

Why? Event **bubbles up** from inner element to outer elements (child → parent → grandparent)

---

stopPropagation() - Stop Bubbling

javascript

```
second.addEventListener("click", function(e){
    e.stopPropagation()  // Stops event from bubbling up
    console.log("second")
})
```

Now when you click "Third":
```
Output:
third
second
(stops here, doesn't reach "first")
```

Event Flow Phases:

1. **Capturing Phase** (top → down)
2. **Target Phase** (the element itself)
3. **Bubbling Phase** (bottom → up) ← **Default**

javascript

```
// Bubbling (default)
element.addEventListener("click", handler)
element.addEventListener("click", handler, false)

// Capturing
element.addEventListener("click", handler, true)
```

Capturing Example:

javascript

```
first.addEventListener("click", function(){
    console.log("first")
}, true)  // Capturing phase

second.addEventListener("click", function(){
    console.log("second")
}, true)

third.addEventListener("click", function(){
    console.log("third")
}, true)
```

When you click "Third":

```
Output (Capturing):
first
second
third
```

Visual Explanation:

```
When clicking "Third":

CAPTURING (top → down):
┌─────────────────┐
│   first         │ ← Executes first (if capturing=true)
│  ┌──────────┐   │
│  │ second   │   │ ← Executes second
│  │  ┌─────┐ │   │
│  │  │third│ │   │ ← TARGET (executes)
│  │  └─────┘ │   │
│  └──────────┘   │
└─────────────────┘

BUBBLING (bottom → up):
┌─────────────────┐
│   first         │ ← Executes last
│  ┌──────────┐   │
```

```
┌─────────────┐
│  second     │  ← Executes second
│  ┌────────┐ │
│  │ third  │ │  ← Executes first (default)
│  └────────┘ │
└─────────────┘
```

📊 Summary: Event Methods

Method	Multiple Functions?	Remove?	Event Object?
Inline `onclick=""`	❌ No	❌ No	⚠️ Manual
`element.onclick`	❌ No	⚠️ Set to null	✅ Yes
`addEventListener()`	✅ Yes	✅ Yes	✅ Yes

🎯 Best Practices:

1. **Always use** `addEventListener()` for better control
2. **Use named functions** if you need to remove listeners
3. **Use** `e.stopPropagation()` carefully (can break functionality)
4. **Check** `e.target` **vs** `e.currentTarget` for delegation
5. **Store timer IDs** to clear them later

javascript

```
var timer = setTimeout(fn, 1000)
clearTimeout(timer)
```

Common Event Use Cases:

1. Form Validation

javascript

```
form.addEventListener("submit", function(e){
    e.preventDefault() // Stop form submission

    if(input.value === ""){
        alert("Field cannot be empty")
    } else {
        form.submit()
    }
})
```

2. Click Counter

javascript

```
var count = 0
btn.addEventListener("click", function(){
    count++
    span.innerHTML = count
})
```

3. Keyboard Shortcuts

javascript

```
document.addEventListener("keydown", function(e){
    if(e.ctrlKey && e.key === "s"){
        e.preventDefault()
        saveDocument()
    }
})
```

4. Hover Effects

javascript

```
div.addEventListener("mouseenter", function(){
    this.style.backgroundColor = "yellow"
})

div.addEventListener("mouseleave", function(){
    this.style.backgroundColor = "white"
})
```

BY. Abdullah Ali

Contact : +201012613453