



**THỰC HỌC – THỰC NGHIỆP**

# **LẬP TRÌNH JAVASCRIPT**

**PRE-INTERMEDIATE JAVASCRIPT**

# Hệ thống bài cũ

---

- Javascript cung cấp đối tượng style trong mỗi element để thay đổi style cho các element.
- Javascript cung cấp cách để chỉ định sự kiện cho element
  - Chỉ định sự kiện bằng HTML
  - Chỉ định sự kiện bằng Javascript
  - Chỉ định sự kiện với event listeners
- Mouse event handlers gồm:
  - Onbclick, onmousedown, onmouseup, onmouseenter, onmouseleave
  - Onmousemove, onmouseout, onmouseover
- Có thể dùng javascript kết hợp mouse event để tạo hiệu ứng rollover, slideshow

# MỤC TIÊU BÀI HỌC

---

- Event target property
- onChange & onBlur
- Key event
- Drag & Drop elements
- Form Submission.
- Pre-intermediate javascript: function & arguments object, js hoisting, strict mode,...

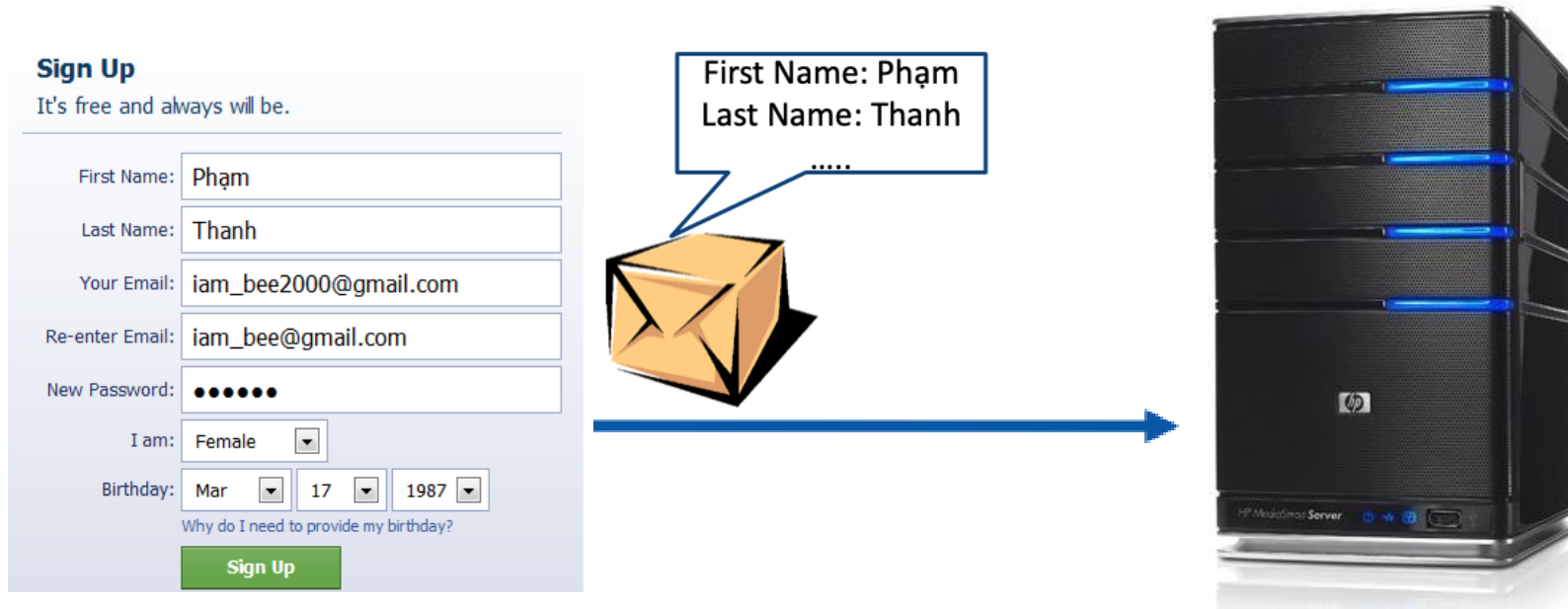
# Form & truy xuất Form

---

- **Nhắc lại bài cũ:** Form và input (text, radio, checkbox, select,...), button..
- **Nhắc lại bài cũ:** có thể truy cập đến các element (node) bằng cách sau
  - Truy xuất các element (node) **bằng ID: `getElementById()`**
  - Truy xuất các element (node) **bằng tên thẻ (Tag name): `getElementsByTagName()`**
  - Truy xuất các element (node) **bằng tên class (Class name): `getElementsByClassName()`**
  - Truy xuất các element (node) với **CSS selector: `querySelector()`, `querySelectorAll()`**

# Form

- Dùng để gửi dữ liệu lên server
- Khi người dùng nhấn vào Button Submit thì dữ liệu sẽ được đóng gói và gửi lên Server
- Form sử dụng phương thức GET hoặc POST để gửi dữ liệu lên Server



# Kiểm tra hợp lệ cho form

---

- Trước đây việc kiểm tra hợp lệ được thực hiện trên server (server side validation)
- Javascript ra đời, thực hiện kiểm tra hợp lệ trên browser (client-side validation) trước khi gửi dữ liệu lên server
- Với một số lượng lớn người truy cập, kiểm tra hợp lệ trên browser sẽ giảm tải cho server
- Chú ý: vẫn phải kiểm tra hợp lệ trên server vì người dùng có thể disable javascript

# Form và DOM

---

- Một số element được định nghĩa thêm thuộc tính name (ví dụ như các điều khiển checkbox, radio,...)
- Có thể sử dụng attribute name để truy cập đến một nhóm các element có cùng giá trị attribute name
- Phân biệt id và name
  - Id là duy nhất, mỗi id đại diện cho 1 element
  - Nhiều phần tử có cùng giá trị của attribute name, mỗi giá trị name đại diện cho một nhóm các phần tử
- Sử dụng phương thức **getElementsByName(name)** để lấy về một mảng các element có cùng thuộc tính name

# Form và DOM

## (Demo truy xuất form: input text field)

```
<form id="my-frm">
  <input type="text" name="firstname" placeholder="First name" />
  <input type="text" name="lastname" placeholder="Last name" />
  <input type="button" onclick="sendInfo()" value="Submit" />
</form>
<script>
  function sendInfo() {
    var inputs = document.getElementById("my-frm").elements;
    var inputByIndex = inputs[0];
    var inputByName = inputs["firstname"];
    message("Welcome " + inputs["firstname"].value + " " + inputs["lastname"].value);
  }
  function message(m) {
    document.getElementById("welcome").innerHTML = m;
  }
</script>
```



# Form và DOM

## (Demo truy xuất form: input checkbox field)

```
<body>
  <div id="welcome">Hi there!</div>
  <input type="checkbox" name="test" value="how"/>
  <input type="checkbox" name="test" value="are"/>
  <input type="checkbox" name="test" value="you"/>
  <input type="checkbox" name="test" value="?" />
  <script type="text/javascript">
    var ckAr = document.getElementsByName("test"); var str = "";
    for (var i = 0; i < ckAr.length; i++) {
      str = str + " " + ckAr[i].value }
    alert(str);
  </script>
</body>
```

# Event target property

---

- **Thuộc tính target của sự kiện**

- Bất cứ khi nào sự kiện được kích hoạt > một biến sự kiện (event variable) sẽ có sẵn.
- Biến sự kiện có rất nhiều thuộc tính, có thể được kiểm tra bằng lệnh:

```
console.dir(event);
```

- Target là thuộc tính của biến sự kiện
- Target là HTML element đã kích hoạt sự kiện

# Demo: Event target property

---

```
<html>
  <head>
    <title>Hi</title>
  </head>
  <script>...
</script>
  <body>
    <button type="button" onclick="triggerSomething()">Click</button>
    <script>
      function triggerSomething() {
        console.dir(event.target);
      }
    </script>
  </body>
</html>
```

# Event target property

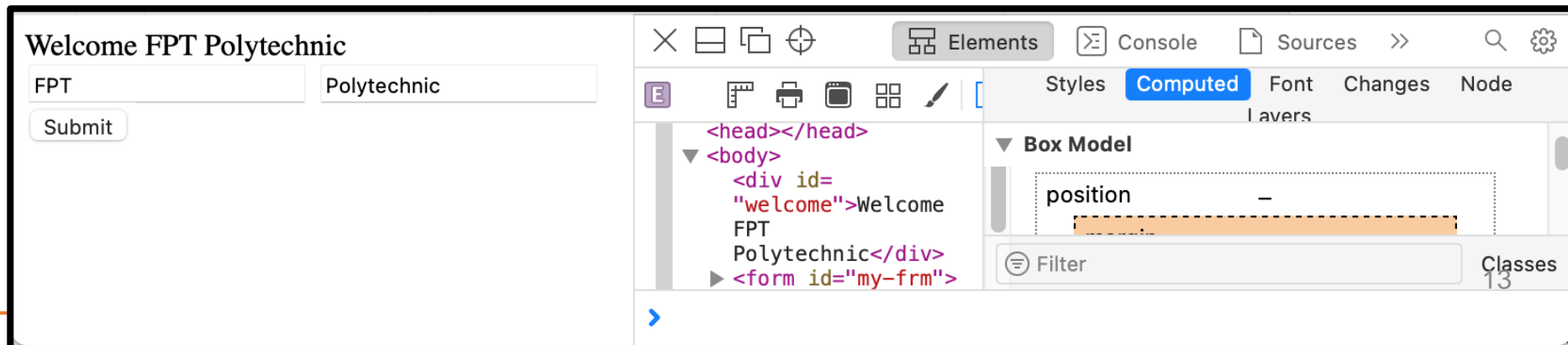
---

- **Thuộc tính target của sự kiện**

- `Event.target` là button element sẽ bao gồm element và các thuộc tính của nó (gồm cả anh em và cha mẹ của nó)
- `Event.target` thường sử dụng trong HTML form (do có nhiều input field và button) . Button trong form thì có cha là form. Thông qua form cha, có thể điều khiển các input nằm trong form cha đó.

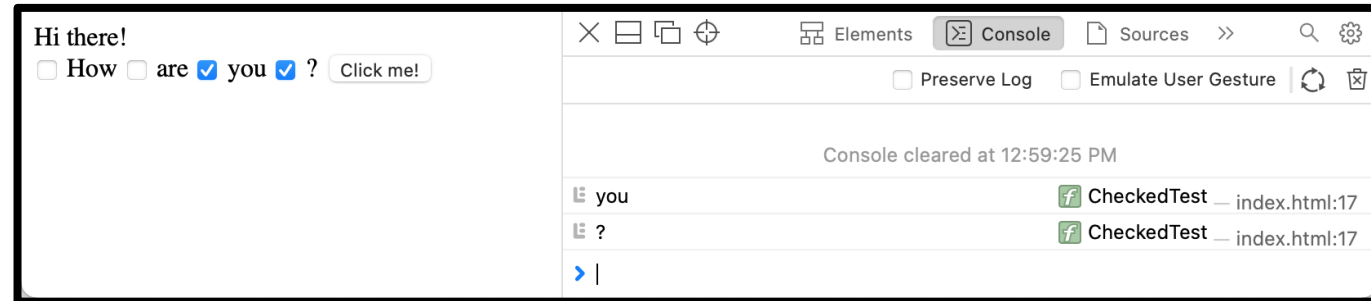
# Truy xuất form (input text field) với event target property

```
<form>
  <input type="text" name="firstname" placeholder="First name" />
  <input type="text" name="lastname" placeholder="Last name" />
  <input type="button" onclick="sendInfo()" value="Submit" />
</form>
<script>
  function sendInfo() {
    let p = event.target.parentElement;
    message("Welcome " + p.firstname.value + " " + p.lastname.value);
  }
  function message(m) {
    document.getElementById("welcome").innerHTML = m;
  }
</script>
```



# Truy xuất form (input checkbox) với event target property

```
<body>
  <div id="welcome">Hi there!</div>
  <form>
    <input type="checkbox" name="test" value="how"/> How
    <input type="checkbox" name="test" value="are"/> are
    <input type="checkbox" name="test" value="you"/> you
    <input type="checkbox" name="test" value="?"/> ?
    <input type="button" onClick="CheckedTest()" value="Click me!"/>
  </form>
  <script type="text/javascript">
    function CheckedTest(){
      let p = event.target.parentElement;
      for(var i=0; i<p.test.length; i++){
        if(p.test[i].checked==true){
          console.log(p.test[i].value);
        }
      }
    }
  </script>
</body>
```



# Onchange , onblur

- **Onchange, onblur** là 2 event thường kết hợp với input
  - **Onchange** được kích hoạt khi một phần tử thay đổi. Ví dụ: giá trị của input thay đổi
  - **Onblur** được kích hoạt khi một đối tượng mất focus. Ví dụ: khi con trỏ đang trong input này và chuyển đến input khác, sự kiện onblur của input đầu tiên được kích hoạt

Hi there!

First Name Changed to FPT

# Demo onchange, onblur

```
<form>
  <input type="text" name="firstname" placeholder="First name" onchange="logEvent()" />
  <input type="text" name="lastname" placeholder="Last name" onblur="logEvent()" />
  <input type="button" onclick="sendInfo()" value="Submit" />
</form>
```

```
<script>
  function logEvent() {
    let p = event.target;
    if (p.name == "firstname") {
      message("First Name Changed to " + p.value);
    } else {
      message("Last Name Changed to " + p.value);
    }
  }
  function sendInfo() {
    let p = event.target.parentElement;
    message("Welcome " + p.firstname.value + " " + p.lastname.value);
  }
  function message(m) {
    document.getElementById("welcome").innerHTML = m;
  }
</script>
```

First Name Changed to FPT

FPT Last name

Submit



# Key event handlers

---

- Sự kiện xảy ra khi người dùng nhấn phím (trên bàn phím)

Sự kiện	Mô tả
onkeydown	Sự kiện xảy ra khi người dùng <b>đang nhấn phím</b>
onkeypress	Sự kiện xảy ra khi người dùng <b>nhấn phím</b>
onkeyup	Sự kiện xảy ra khi người dùng <b>buông/nhả phím</b>

# Demo key event handlers

Hi there!  
JavaScript is fun!

Hi there!  
Number: false

Hi there!  
Not a number: true

```
<div id="welcome">Hi there!</div>
<div id="wrapper">JavaScript is fun!</div>
<input type="text" name="myNum1" onkeypress="return numCheck()" onpaste="return false">
<input type="text" name="myNum2" onkeypress="return numCheck2()" onpaste="return false">
<script>
  function numCheck() {
    message("Number: " + !isNaN(event.key));
    return !isNaN(event.key);
  }
  function numCheck2() {
    message("Not a number: " + isNaN(event.key));
    return isNaN(event.key);
  }
  function message(m) {
    document.getElementById('wrapper').innerHTML = m;
  }
</script>
```

# Form submission

---

- Mỗi form có một hoặc nhiều button Submit
- Sự kiện `onsubmit` của form
  - Được kích hoạt khi người dùng nhấn vào button Submit
  - Nếu `onsubmit` có giá trị trả về là `True`, dữ liệu được gửi lên server
  - Nếu `onsubmit` có giá trị trả về là `False`, dữ liệu không được gửi lên server

# Form submission

First name

Last name

Age

submit

Need a first name!!

First name

Last name

Age

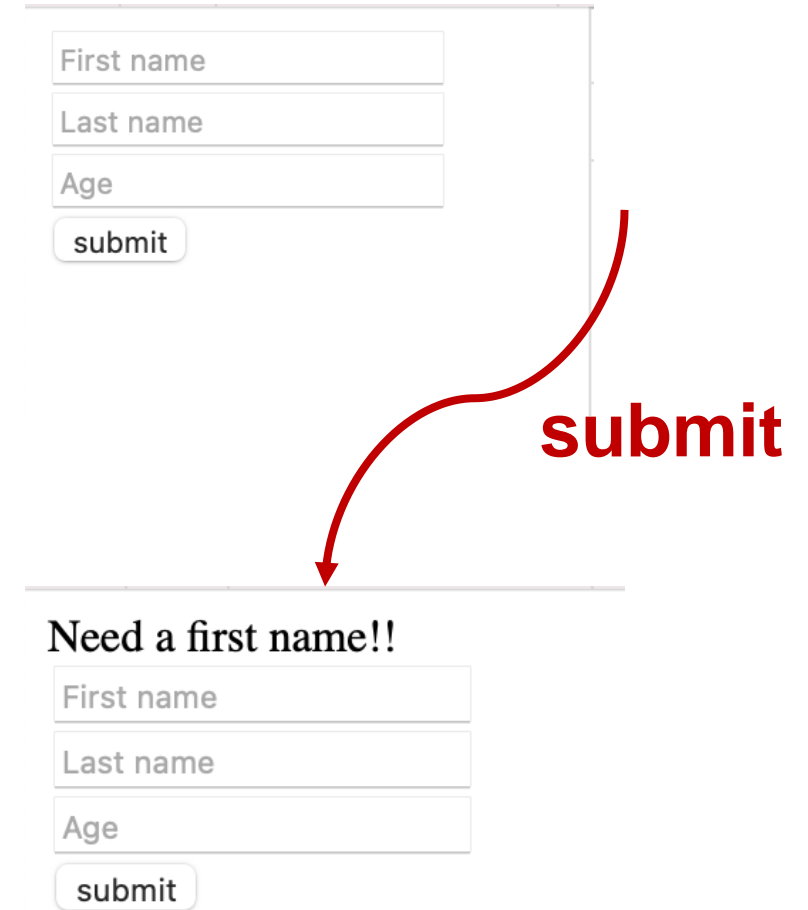
submit

```
<div id="wrapper"></div>
<form action="anotherpage.html" method="get" onsubmit="return valForm()">
  <input type="text" id="firstName" name="firstName" placeholder="First name" />
  <input type="text" id="lastName" name="lastName" placeholder="Last name" />
  <input type="text" id="age" name="age" placeholder="Age" />
  <input type="submit" value="submit" />
</form>
```

# Demo form submission

```
<script>
function valForm() {
    var p = event.target.children;
    if (p.firstName.value == "") {
        message("Need a first name!!");
        return false;
    }
    if (p.lastName.value == "") {
        message("Need a last name!!");
        return false;
    }
    if (p.age.value == "") {
        message("Need an age!!");
        return false;
    }
    return true;
}

function message(m) {
    document.getElementById("wrapper").innerHTML = m;
}
</script>
```



The diagram illustrates a form submission process. It shows a form with three input fields: "First name", "Last name", and "Age", followed by a "submit" button. A red arrow points from the "submit" button to a message box that displays the error message "Need a first name!!". The message box also contains the same three input fields and a "submit" button, indicating that the form is re-rendered with the error message.

# Drag và drop elements

---

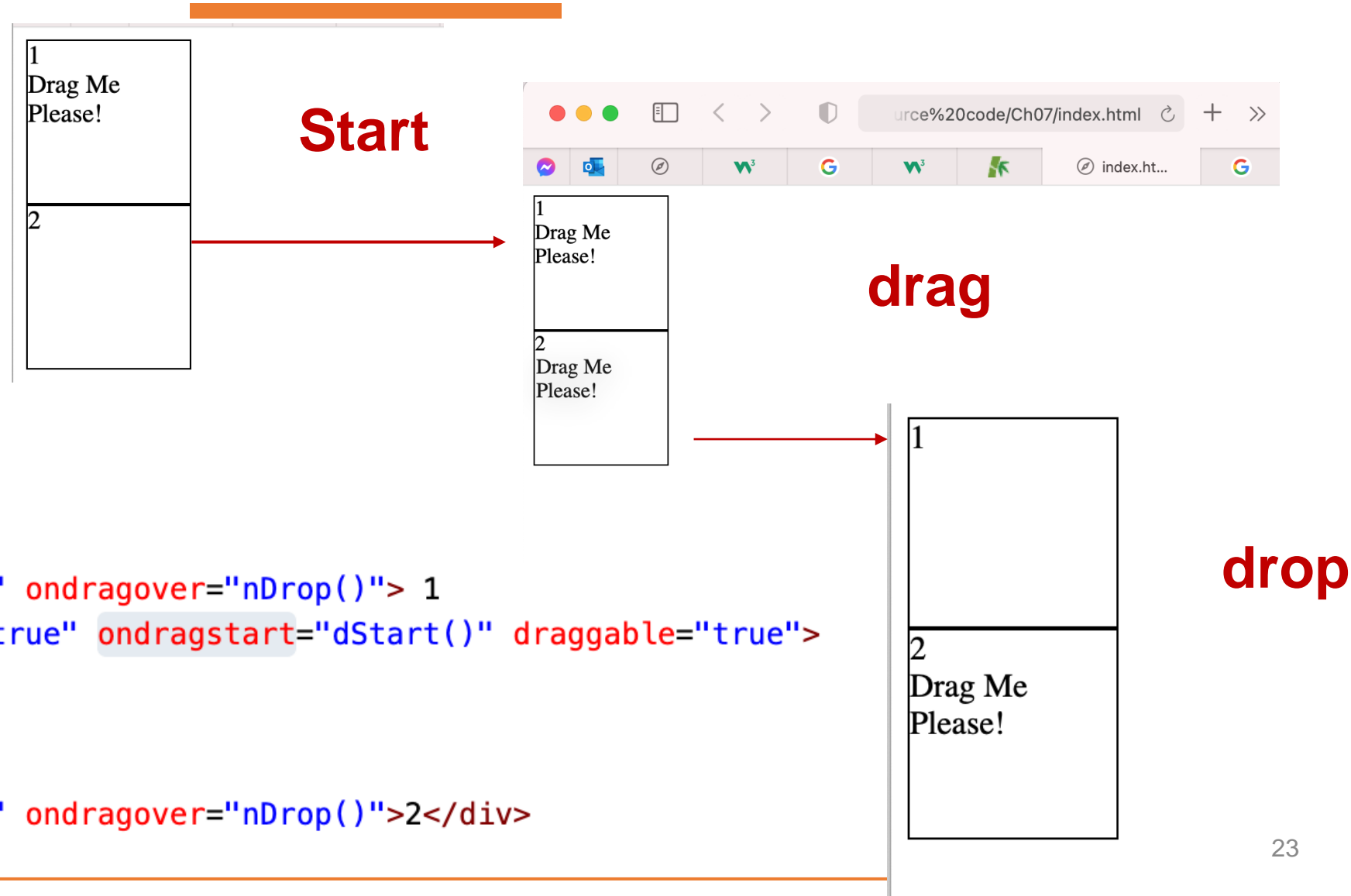
- `event.preventDefault`: ngăn chặn cách xử lý mặc định của trình duyệt khi xảy ra sự kiện

Sự kiện	Mô tả
<code>ondrag</code>	Xảy ra khi chọn 1 element (text) để kéo và đang kéo
<code>ondrop</code>	Xảy ra khi 1 element/text (cho phép kéo) được thả xuống vị trí hợp lệ
<code>ondragstart</code>	Xảy ra khi người dùng kéo 1 element
<code>ondragover</code>	Xảy ra khi phần tử được kéo vượt quá mục tiêu thả

# Demo drag và drop elements

```
<style>
  .box {
    width: 100px;
    height: 100px;
    border: 1px solid black;
    background-color: white;
  }
  .red {
    background-color: red;
  }
</style>
```

```
<div class="box" ondrop="dDrop()" ondragover="nDrop()"> 1
  <div id="dragme" draggable="true" ondragstart="dStart()" draggable="true">
    Drag Me Please!
  </div>
</div>
<div class="box" ondrop="dDrop()" ondragover="nDrop()">2</div>
```



# Demo drag và drop elements

---

```
<script>
    let holderItem;
    function dStart() {
        holderItem = event.target;
    }
    function nDrop() {
        event.preventDefault();
    }
    function dDrop() {
        event.preventDefault();
        if (event.target.className == "box") {
            event.target.appendChild(holderItem);
        }
    }
</script>
```



# Functions và đối tượng arguments

---

- Javascript xử lý các đối số trong hàm bằng cách thêm vào một đối tượng tùy chỉnh gọi là arguments
- arguments:
  - hoạt động giống mảng.
  - Sử dụng arguments thay vì sử dụng tên của tham số
- Ví dụ:

```
function test(a, b, c) {  
    console.log("first:", a, arguments[0]);  
    console.log("second:", b, arguments[1]);  
    console.log("third:", c, arguments[2]);  
}  
test("fun", "js", "secrets");
```

# Javascript hoisting

---

- Hoisting là default behavior (hành vi mặc định) của javascript là di chuyển tất cả các declaration (khai báo) lên đầu phạm vi hiện tại (lên đầu tập lệnh hiện tại hoặc hàm hiện tại)
- Hoisting (đối với nhiều developers) là hành vi không xác định thường bị bỏ qua nhưng nếu developers không hiểu về lưu trữ, các chương trình có thể chứa lỗi
- Để tránh lỗi, hãy luôn khai báo tất cả các biến ở đầu mọi phạm vi (scope)

# DEMO javascript hoisting

---

```
var x = 5; // Initialize x
```

```
elem = document.getElementById("demo"); // Find an element  
elem.innerHTML = x + " " + y;           // Display x and y
```

```
var y = 7; // Initialize y
```

- Hoisting có nghĩa là javascript tự động khai báo biến y trên đầu phạm vi như các dòng code phía dưới đây

```
var x = 5; // Initialize x  
var y;     // Declare y
```

```
elem = document.getElementById("demo"); // Find an element  
elem.innerHTML = x + " " + y;           // Display x and y
```

```
y = 7;     // Assign 7 to y
```

# Demo javascript hoisting

---

- Chú ý: nếu khai báo biến y với let sẽ xảy ra lỗi

```
var x = 5; // Initialize x
```

```
elem = document.getElementById("demo"); // Find an element  
elem.innerHTML = x + " " + y;           // Display x and y
```

```
let y = 7; // Initialize y
```

# Sử dụng strict mode

---

- strict mode (ES5): không cho phép sử dụng các biến nếu chúng không được khai báo
- Khai báo “**use strict**”; bắt đầu 1 script hoặc 1 hàm
- Ví dụ

```
"use strict";  
x = 3.14;           // This will cause an error because x is not  
declared
```

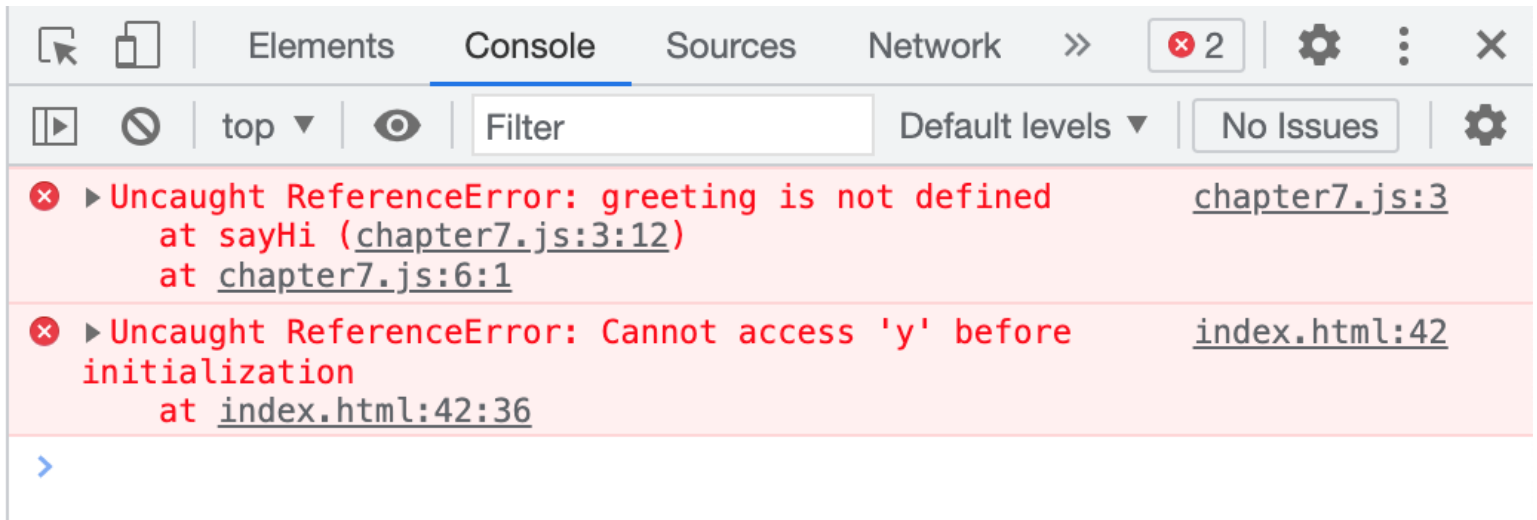
---

Hoặc

```
"use strict";  
myFunction();  
  
function myFunction() {  
    y = 3.14;        // This will also cause an error because y is not  
declared  
}
```

# Demo sử dụng strict mode

```
"use strict";  
function sayHi() {  
    greeting = "Hello!";  
    console.log(greeting);  
}  
sayHi();
```



# Tổng kết bài học

---

- Event target property
- onChange & onBlur
- Key event
- Drag & Drop elements
- Form Submission.
- Pre-intermediate javascript: function & arguments object, js hoisting, strict mode,...

*Thank  
you!*