



## **BÀI 7: JQUERY TRAVERSE (PHẦN 1)**

- ⊙ Tìm kiếm các phần tử bởi (index)
- ⊙ Lọc các phần tử trong jQuery
- ⊙ Xác định vị trí các phần tử con
- ⊙ Các phương thức DOM Traversing
- ⊙ Từ khóa Void trong JavaScript
- ⊙ Xử lý lỗi với try - catch



- ❑ jQuery là một công cụ vô cùng mạnh mẽ. Nó cung cấp các phương thức đa dạng để truy cập DOM (DOM Traversal Method), giúp chúng ta chọn các phần tử trong một tài liệu một cách ngẫu nhiên hoặc theo phương thức liên tục.
- ❑ Hầu hết DOM Traversal Method không sửa đổi đối tượng jQuery và chúng được sử dụng để lọc các phần tử của chúng từ một tài liệu trên cơ sở các điều kiện đã cho.

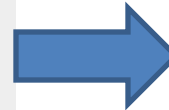
## Code

```
<html>
  <head>
    <title>The JQuery Example</title>
  </head>

  <body>

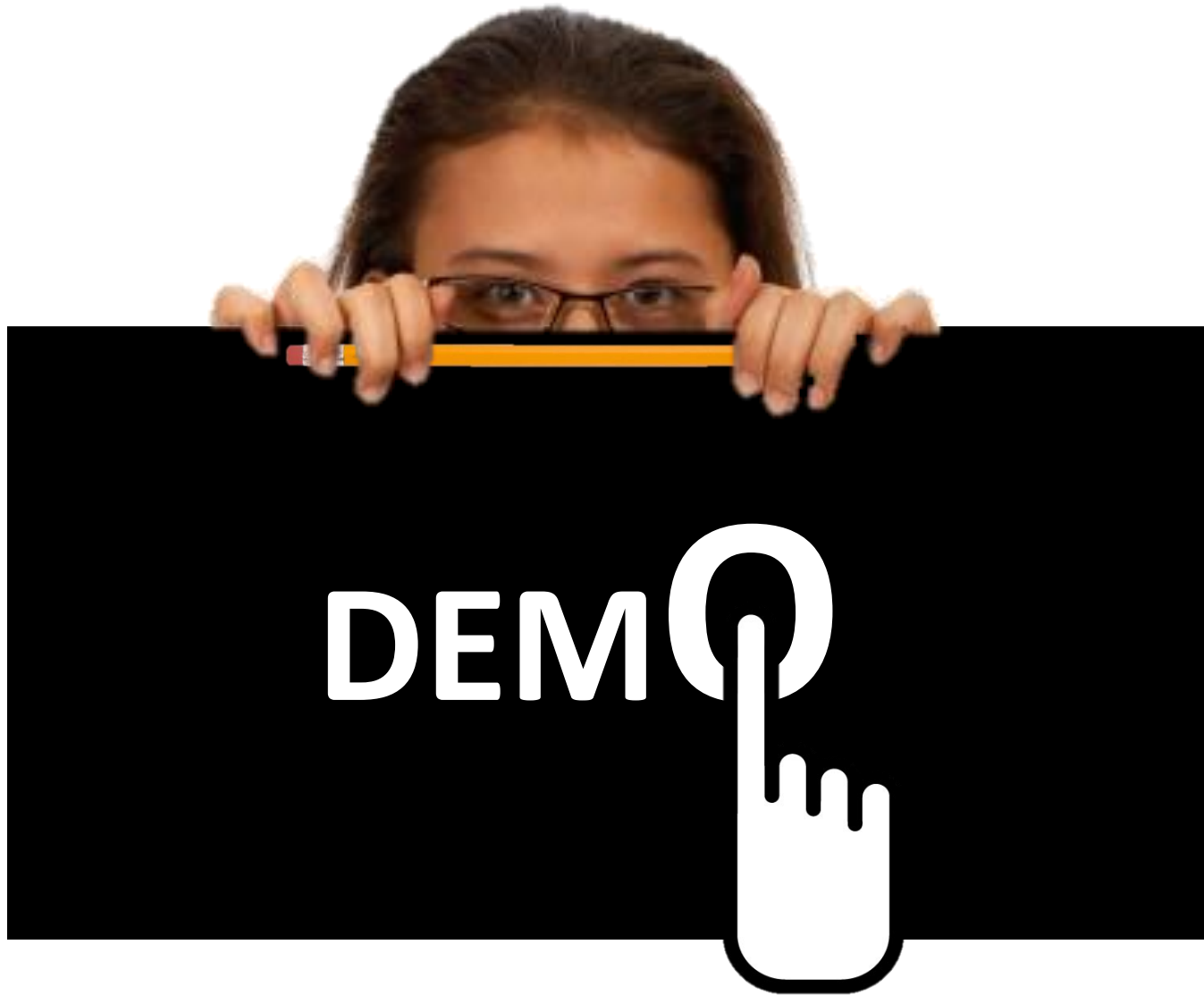
    <div>
      <ul>
        <li>list item 1</li>
        <li>list item 2</li>
        <li>list item 3</li>
        <li>list item 4</li>
        <li>list item 5</li>
        <li>list item 6</li>
      </ul>
    </div>

  </body>
</html>
```



Nó sẽ tạo kết quả sau:

- list item 1
- list item 2
- list item 3
- list item 4
- list item 5
- list item 6



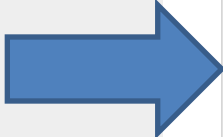
- ❑ Ở trên, mỗi list có một index riêng của nó, và có thể được xác định vị trí bởi sử dụng phương thức **eq(index)** như ví dụ dưới.
- ❑ Mỗi phần tử con bắt đầu index của nó từ 0, vì thế, list item 2 sẽ được truy cập bởi sử dụng **\$("li").eq(1)** và tiếp tục như thế.
- ❑ Ví dụ:

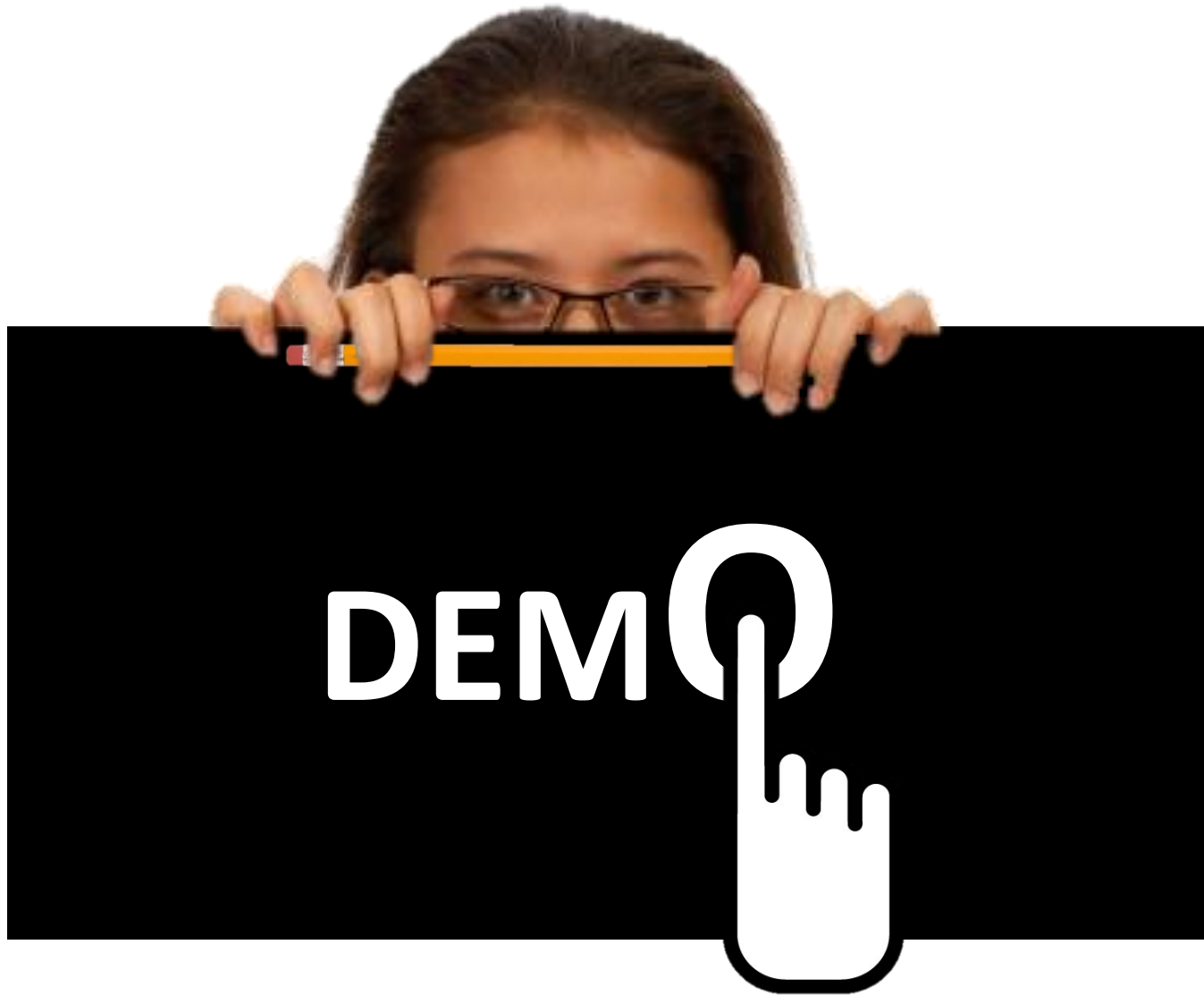
```
<html>
  <head>
    <title>The JQuery Example</title>
    <script type="text/javascript" src="http://ajax.googleapis.com/ajax/libs/jquery
    </script>

    <script type="text/javascript" language="javascript">
      $(document).ready(function() {
        $("li").eq(2).addClass("selected");
      });
    </script>

    <style>
      .selected { color:red; }
    </style>
  </head>
  <body>
    <div>
      <ul>
        <li>list item 1</li>
        <li>list item 2</li>
        <li>list item 3</li>
        <li>list item 4</li>
        <li>list item 5</li>
        <li>list item 6</li>
      </ul>
    </div>
  </body>
</html>
```

Nó sẽ cho kết quả sau:

- 
- list item 1
  - list item 2
  - **list item 3**
  - list item 4
  - list item 5
  - list item 6





- ❑ Phương thức **filter( selector )** có thể được sử dụng để lọc ra tất cả các phần tử từ tập hợp các phần tử được so khớp mà không so khớp với Selector đã cho. *selector* có thể được viết bởi sử dụng bất kỳ cú pháp Selector nào.
- ❑ Ví dụ đơn giản sau áp dụng màu tới các list mà liên kết với class là *middle*

## Code

```
<html>
  <head>
    <title>The JQuery Example</title>
    <script type="text/javascript" src="http://ajax.googleapis.com/ajax/libs/jquery/
    </script>

    <script type="text/javascript" language="javascript">
      $(document).ready(function() {
        $("li").filter(".middle").addClass("selected");
      });
    </script>

    <style>
      .selected { color:red; }
    </style>

  </head>

  <body>

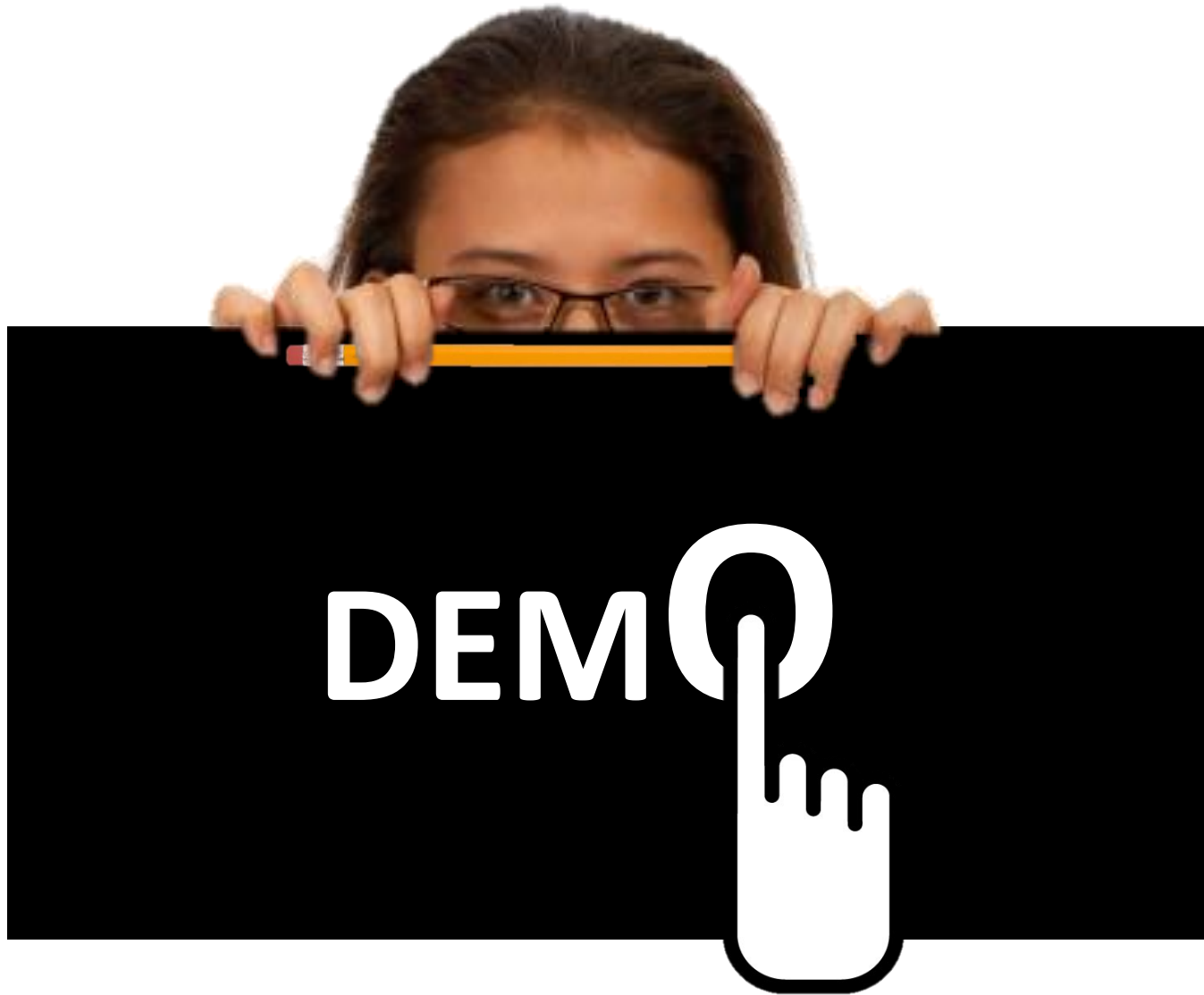
    <div>
      <ul>
        <li class="top">list item 1</li>
        <li class="top">list item 2</li>
        <li class="middle">list item 3</li>
        <li class="middle">list item 4</li>
        <li class="bottom">list item 5</li>
        <li class="bottom">list item 6</li>
      </ul>
    </div>

  </body>
```

❑ Kết quả:

Nó sẽ cho kết quả sau:

- list item 1
- list item 2
- list item 3
- list item 4
- list item 5
- list item 6



- ❑ Phương thức **find( selector )** có thể được sử dụng để xác định vị trí tất cả phần tử con của một loại phần tử cụ thể. *selector* có thể được viết bởi sử dụng bất kỳ cú pháp selector nào.
- ❑ Ví dụ đơn giản sau chọn tất cả phần tử `<span>` có sẵn bên trong các phần tử `<p>` khác nhau.

```
<html>
  <head>
    <title>The JQuery Example</title>
    <script type="text/javascript" src="http://ajax.googleapis.com/ajax/libs/jquery/
    </script>

    <script type="text/javascript" language="javascript">
      $(document).ready(function() {
        $("p").find("span").addClass("selected");
      });
    </script>

    <style>
      .selected { color:red; }
    </style>

  </head>

  <body>
    <p>This is 1st paragraph and <span>THIS IS RED</span></p>
    <p>This is 2nd paragraph and <span>THIS IS ALSO RED</span></p>
  </body>

</html>
```

❑ Kết quả:

Nó sẽ cho kết quả sau:

This is 1st paragraph and THIS IS RED

This is 2nd paragraph and THIS IS ALSO RED

STT	Phương thức & Miêu tả
1	<b>add( selector )</b> Thêm nhiều phần tử, đã so khớp bởi selector đã cho, tới tập hợp các phần tử được so khớp.
2	<b>andSelf( )</b> Thêm sự chọn lọc trước tới sự chọn lọc hiện tại
3	<b>children( [selector])</b> Nhận một tập hợp các phần tử chứa tất cả các phần tử con trực tiếp duy nhất của mỗi một trong tập hợp các phần tử được so khớp.
4	<b>closest( selector )</b> Nhận một tập hợp các phần tử chứa phần tử cha gần nhất mà so khớp selector đã cho, bao gồm phần tử bắt đầu
5	<b>contents( )</b> Tìm tất cả <i>node</i> con bên trong các phần tử được so khớp (bao gồm cả các text node), hoặc nội dung tài liệu, nếu phần tử là một Iframe.



6	<b>end( )</b> Biến đổi hoạt động phá hủy (destructive) gần đây nhất, thay đổi tập hợp các phần tử về trạng thái trước đó.
7	<b>find( selector )</b> Tìm kiếm các phần tử con mà so khớp với selector đã cho.
8	<b>next( [selector] )</b> Nhận một tập hợp các phần tử chứa anh (em) kế tiếp duy nhất của mỗi phần tử trong tập hợp phần tử đã cho.
9	<b>nextAll( [selector] )</b> Tìm tất cả các phần tử anh em kế sau phần tử hiện tại
10	<b>offsetParent( )</b> Trả về một tập hợp jQuery với phần tử cha đã xác định vị trí của phần tử được so khớp đầu tiên
11	<b>parent( [selector] )</b> Nhận cha trực tiếp của một phần tử. Nếu được gọi trên một tập hợp các phần tử, phương thức parent trả về một tập hợp các phần tử cha trực tiếp duy nhất

12	<b>parents( [selector] )</b> Nhận một tập hợp các phần tử chứa tổ tiên (ancestor) của tập hợp các phần tử đã so khớp (ngoại trừ các phần tử root)
13	<b>prev( [selector] )</b> Nhận một tập hợp các phần tử chứa phần tử anh em ở trước duy nhất của mỗi phần tử trong tập hợp các phần tử đã so khớp.
14	<b>prevAll( [selector] )</b> Tìm tất cả phần tử anh em ở trước phần tử hiện tại
15	<b>siblings( [selector] )</b> Nhận một tập hợp phần tử chứa tất cả phần tử anh em duy nhất của mỗi phần tử trong tập hợp các phần tử đã cho



## **BÀI 7: JQUERY TRAVERSE(PHẦN 2)**

❑ **void** là một từ khóa quan trọng trong JavaScript mà có thể được sử dụng như là một toán tử một ngôi xuất hiện trước toán hạng đơn của nó, mà có thể trong bất kỳ kiểu nào. Toán tử này xác định một biểu thức để được ước lượng mà không trả về một giá trị nào.

❑ Cú pháp

```
<head>

  <script type="text/javascript">
    <!--
      void func()
      javascript:void func()

      or:

      void(func())
      javascript:void(func())
    //-->
  </script>

</head>
```

- ❑ Sử dụng phổ biến nhất của toán tử này là trong một Client-Side **javascript**:URL, nó cho phép ước lượng một biểu thức cho các tác động thứ yếu của nó (side-effects) mà trình duyệt không hiển thị giá trị của biểu thức đã ước lượng.
- ❑ Ở đây biểu thức **alert ('Warning!!!')** được ước lượng nhưng nó không được tải trở lại trong tài liệu hiện tại.

## ❑ Ví dụ 1:


```
<html>
  <head>

    <script type="text/javascript">
      <!--
      //-->
    </script>

  </head>
  <body>

    <p>Click the following, This won't react at all...</p>
    <a href="javascript:void(alert('Warning!!!'))">Click me!</a>

  </body>
</html>
```



## Kết quả

Click the following, This won't react at all...

[Click me!](#)

## ❑ Ví dụ 2

Bạn xem ví dụ sau. Link sau sẽ không là gì bởi vì biểu thức "0" không có tác động trong JavaScript. Ở đây biểu thức "0" được ước lượng, nhưng nó không được tải trở lại trong tài liệu hiện tại.

```
<html>
  <head>

    <script type="text/javascript">
      <!--
      //-->
    </script>

  </head>
  <body>

    <p>Click the following, This won't react at all...</p>
    <a href="javascript:void(0)">Click me!</a>

  </body>
</html>
```



### Kết quả

Click the following, This won't react at all...

[Click me!](#)

## ❑ Ví dụ 3

Cách sử dụng khác của **void** là chủ yếu để tạo giá trị **undefined** như sau:

```
<html>
  <head>

    <script type="text/javascript">
      <!--
        function getValue(){
          var a,b,c;

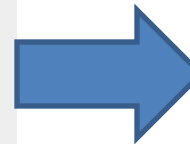
          a = void ( b = 5, c = 7 );
          document.write('a = ' + a + ' b = ' + b + ' c = ' + c );
        }
      <!-->
    </script>

  </head>

  <body>
    <p>Click the following to see the result:</p>

    <form>
      <input type="button" value="Click Me" onclick="getValue();" />
    </form>

  </body>
</html>
```

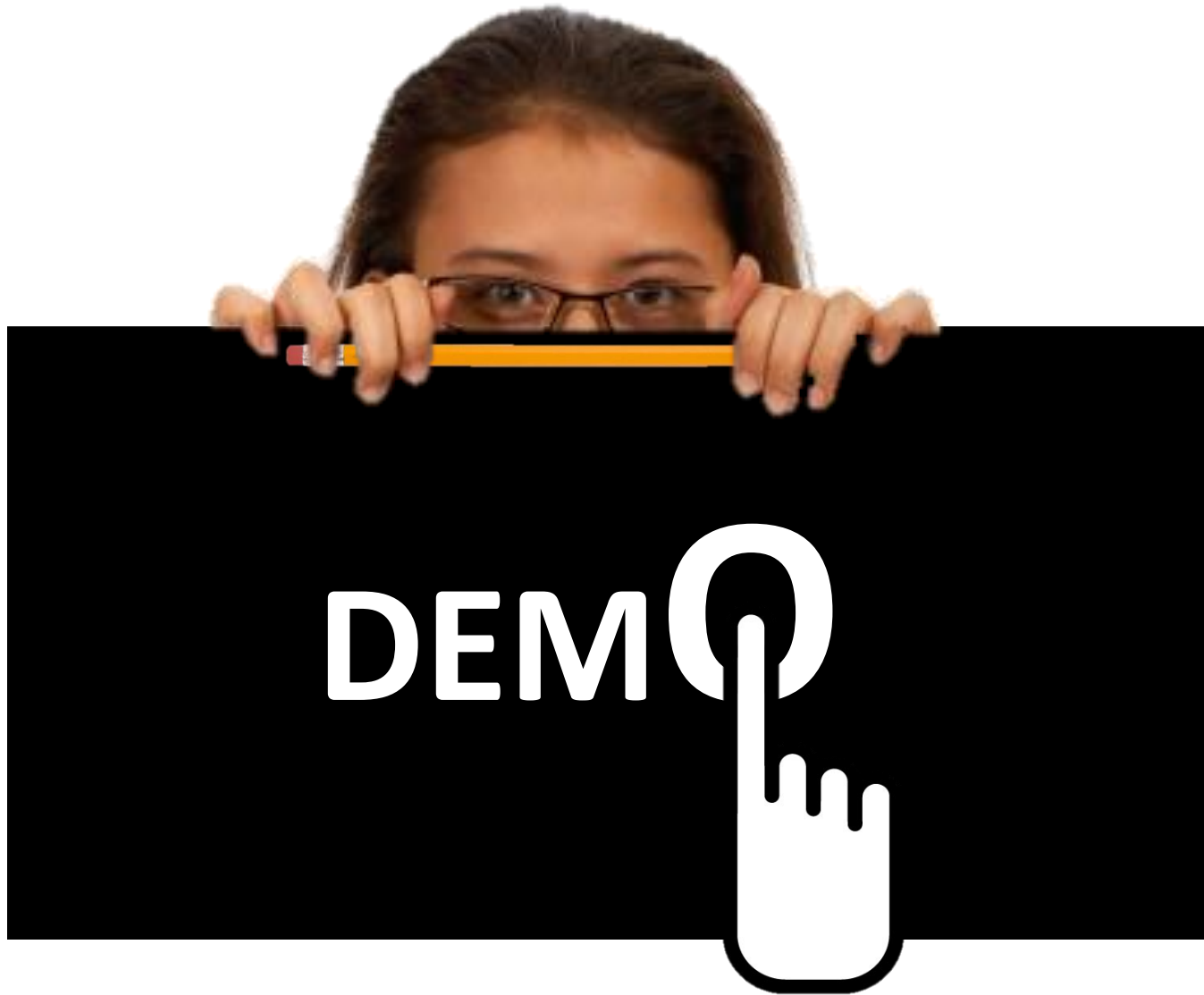


Kết quả

Click the following to see the result:

Click Me





- ❑ Việc quản lý lỗi và xuất thông báo lỗi trong NodeJS thực chất cũng là cách viết trong Javascript.
- ❑ Try - Catch là cấu trúc không còn xa lạ gì trong các ngôn ngữ lập trình như C#, và trong Javascript cũng cung cấp cho chúng ta cú pháp này để xử lý và xuất ra thông báo lỗi.

```
try {  
    // Quảng lỗi ra  
    throw("Nội dung lỗi");  
} catch (e){  
    // Đón nhận lỗi và in ra  
    // Vị trí này chỉ chạy khi ở try có quảng lỗi hoặc ở try  
    // sử dụng sai cú pháp ...  
    console.log(e.message);  
} finally{  
    // Cuối cùng chạy cái này  
    // Luôn luôn chạy sau cùng  
    console.log('End of try catch');  
}
```

❑ Như vậy luồng chạy của chương trình như sau:

- ✓ **Bước 1:** Thực hiện trong try.
- ✓ **Bước 2:** Nếu trong try xuất hiện lỗi thì nhảy sang catch
- ✓ **Bước 3:** Cuối cùng nhảy xuống finally

❑ Như vậy vị trí finally sẽ luôn luôn được thực thi và sẽ thực thi cuối cùng. Với finally có thể sử dụng hoặc không đều được. Riêng trong catch sẽ có một tham số truyền vào và tham số này sẽ chứa thông tin của lỗi và ta sử dụng biến này để lấy message.

- ❑ Ví dụ: Sử dụng biến chưa định nghĩa, nếu bình thường thì chương trình bị dừng nhưng do ta sử dụng try - catch nên chương trình vẫn hoạt động bình thường.

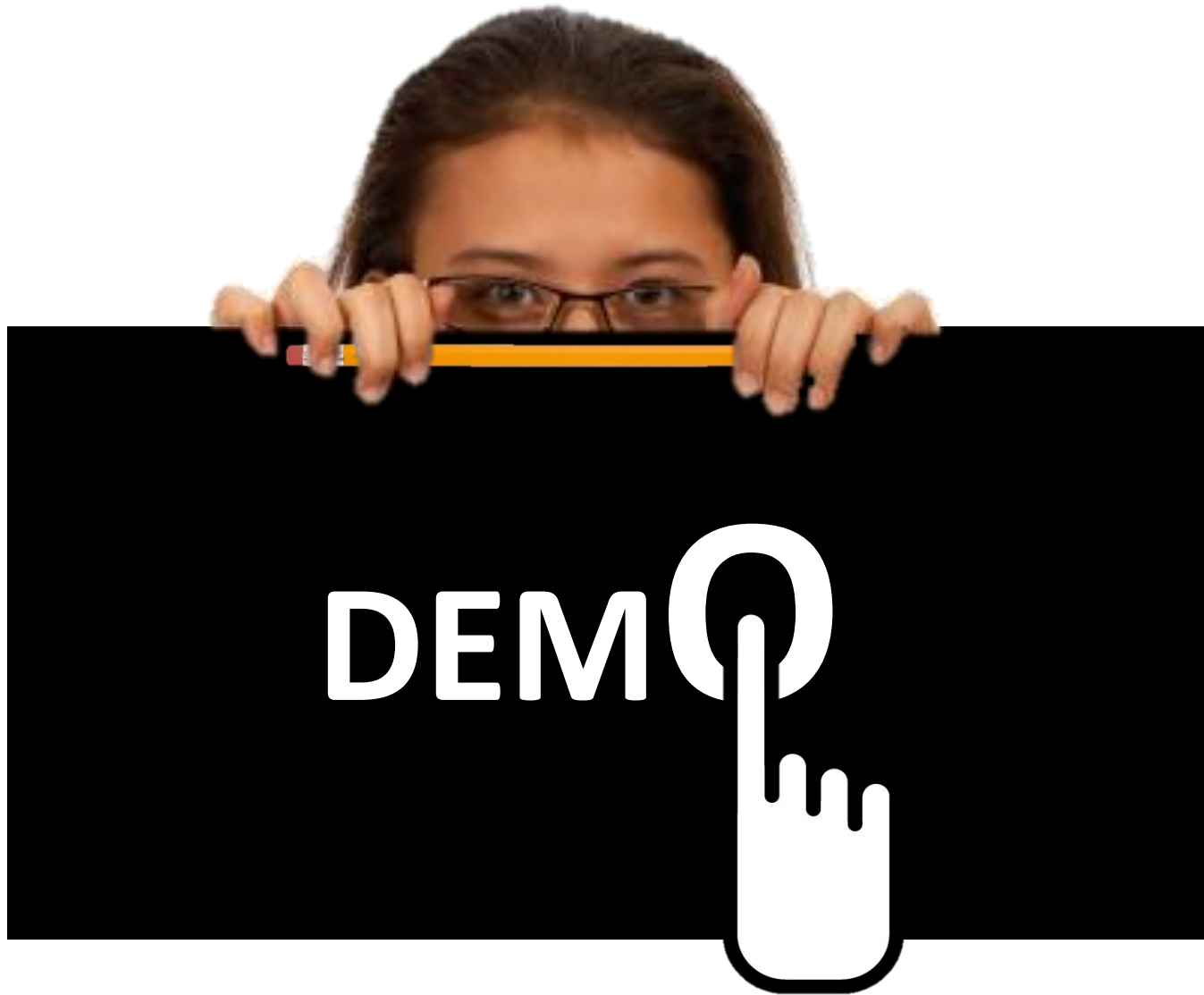
```
1  try {  
2      // Sử dụng biến message chưa được định nghĩa  
3      console.log(message);  
4  } catch (e){  
5      console.log(e.message);  
6  }
```

- ❑ **Ví dụ:** Sử dụng sai cú pháp nhưng chương trình vẫn chạy

```
1  try {  
2      fadsfas  
3      fasdfas  
4      fsda  
5  } catch (e){  
6      console.log(e.message);  
7  } finally{  
8      console.log('End');  
9  }
```

- ❑ **Ví dụ:** Sử dụng `throw new Error('message')` để xuất thông báo lỗi.

```
1  var domain = 'qa.freetuts.net'
2
3  try {
4      if (domain !== 'freetuts.net'){
5          throw new Error('Domain nay khong phai la trang chu');
6      }
7  } catch (e){
8      console.log(e.message);
9  } finally{
10     console.log('End');
11 }
```



- ☑ Tìm kiếm các phần tử bởi (index)
- ☑ Lọc các phần tử trong jQuery
- ☑ Xác định vị trí các phần tử con
- ☑ Các phương thức DOM Traversing
- ☑ Từ khóa Void trong JavaScript
- ☑ Xử lý lỗi với try - catch





**Cảm ơn**