EBU6501 – Middleware

Week 3, Day 5: JavaScript Attributes

Gokop Goteng & Ethan Lau

JavaScript Attributes

- **Object Attributes**
  - Every object has an associated **prototype**, **class** and **extensible** attributes
- **Prototype attributes**
  - This **specifies** the **object** from which it **inherits properties**
  - Prototype attribute is set when an object is created
  - E.g. var p={x:1}; // define a prototype object
  - Var o=Object.create(p); // create an object with that prototype
  - P.isPrototypeOf(o); // => true: o inherits from p
  - Object.prototype.isPrototypeOf(o); // => true: p inherits from Object.prototype
- **Class Attributes**
  - An object’s class attribute is a string that **provides information** about the **type of of the object**.
  - The default “toString()” method which is inherited from Object.prototype returns a string of the form [object class]

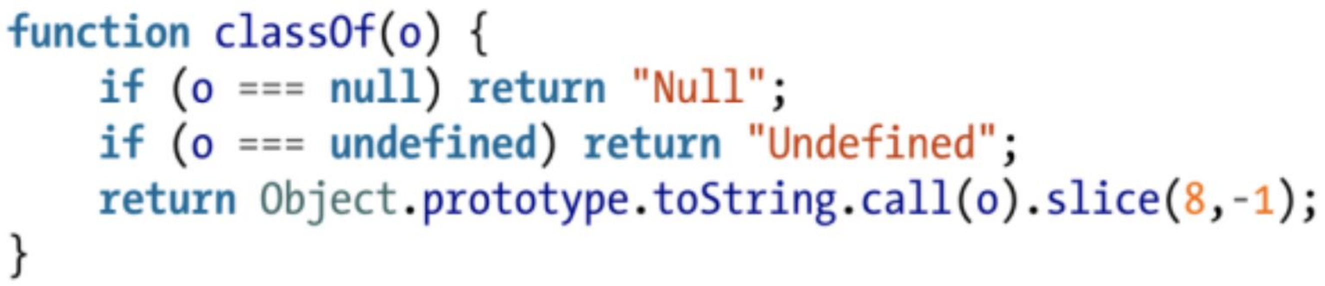
Queen Mary
University of London

JavaScript Attributes

- **Class Attributes**
  - To obtain the class of an object, **invoke the toString()** method on it, and extract the **eighth through the second to last characters** of the returned string (the slice method).

Example:

A function that returns the class of any object that you pass it



```
function classOf(o) {
    if (o === null) return "Null";
    if (o === undefined) return "Undefined";
    return Object.prototype.toString.call(o).slice(8,-1);
}
```

Queen Mary
University of London

JavaScript Attributes

More example:

```
var getType = function (elem) {
        return Object.prototype.toString.call(elem);
};
if (getType(person) === '[object Object]') {
        person.getName();
};
```

Invoke the toString() with slice(8,-1)

```
var getType = function (elem) {
        return Object.prototype.toString.call(elem).slice(8, -1);
};
var isObject = function (elem) {
        return getType(elem) === 'Object'; };
if (isObject(person)) {
        person.getName();
}
```

JavaScript Attributes

- **Class Attributes**
  - Use Function.call() method that defines the class of any object you passed
  - The classof() function works for any JavaScript value such as numbers, strings and Booleans.
  - Examples
    - Classof(null); // => “Null”
    - Classof(2); // => “Number”
    - Classof(false); // => “Boolean”
    - Function f(); //=> “Window”
- **Extensible Attributes**
  - This **specifies** whether **new properties** can be **added to the object or not**
  - Some versions of JavaScript allows extension by default
  - The purpose of extensible attribute is to **lock-down objects** into a known state and **prevent outside tempering**

Queen Mary
University of London

JavaScript Pattern Matching with Regular Expressions

- A **regular expression (RegExp)** is an **object** that describes a **pattern** of **characters**
- The JavaScript RegExp class represents regular expressions
- In JavaScript regular expressions are represented by RegExp objects
- RegExp objects may be created with the **RegExp()** constructor
- used to **match character** combinations in strings
- They are often also created by using special literal syntax
- Example 1:

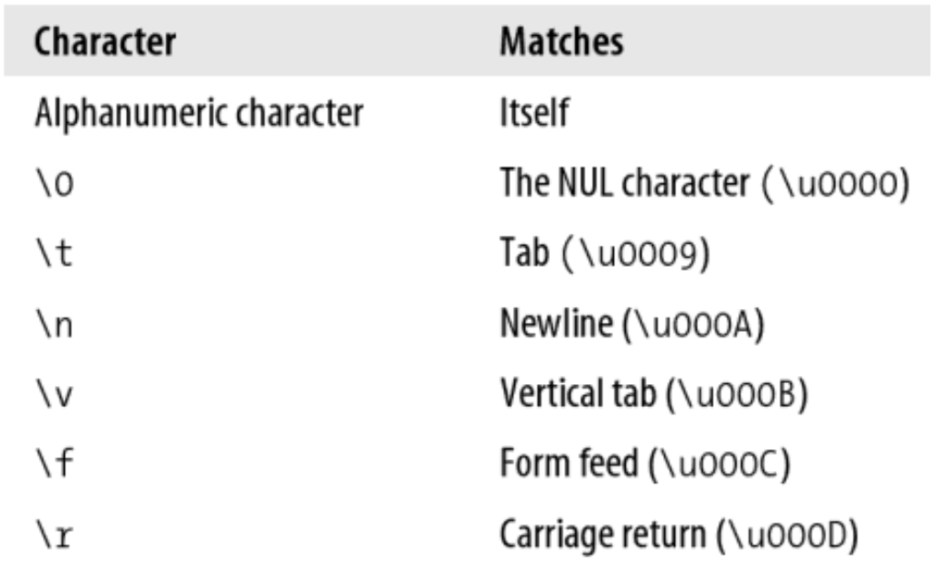
  var re = /ab+c/; Then with RegExp(): var re = new RegExp('ab+c’);
- Example 2: $ matches any string that ends with the letter “s”

  Using RegExp() object, the example becomes “var pattern=new RegExp(“t$”);”

  Note: “t$” does not match the ‘t' in ”eater", but does match it in "eat".

JavaScript Pattern Matching with Regular Expressions

- **Literal Characters**
  - This consists of using all alphabetic characters, digits and certain non-alphabetic characters to **match patterns**. It uses backslash (\), forward-slash (/) and other characters.
  - Examples of regular expression literals



Character	Matches
Alphanumeric character	Itself
\0	The NUL character (\u0000)
\t	Tab (\u0009)
\n	Newline (\u000A)
\v	Vertical tab (\u000B)
\f	Form feed (\u000C)
\r	Carriage return (\u000D)

JavaScript Pattern Matching with Regular Expressions

- **Character Classes**
  - **Individual literal characters** can be **combined** into **character classes** by placing them within **square brackets**
  - A character class may **match** any one character that is contained within it
  - E.g. the regular expression **/[abc]/** matches any of the characters a,b or c
  - Similarly, /[^abc]/ means any character except a, b and c. The character ^ is a negation character
  - You use a hyphen (-) to indicate a range of characters. For example /[a-z]/ means a to z
  - What does **/[a-zA-Z0-9]/** mean?
  - There are escape characters such as \s (matches space character, tab character and any white space character),

Queen Mary
University of London

JavaScript Pattern Matching with Regular Expressions

- **Character Classes**
  - Examples of regular expression classes are
    - […] //any one character **within** the brackets
    - [^…] //any one character **NOT** within the brackets
    - . // any character except newline
    - \d //any ASCII digit, equivalent to [0-9]
- **Repetition**
  - You can **repeat characters**. For example /\d\d/ means two digits and /\d\d\d\d/ means four digits
  - You can use + to mean one or more occurrences of previous pattern
  - Examples of repetition characters
    - {n,m} // match the previous characters at least n times but no more than m times
    - {n,} //match the previous item n or more times
    - {n} //match exactly n times the previous item
    - What is + and *?
      - {1,} and {0,}

JavaScript Pattern Matching with Regular Expressions

- **Repetition**
  - Some more examples
    - /\d{2,4}/ //match between 2 and 4 digits
    - /\w{3}\d?/ // match exactly 3 word characters and an optional digit
    - /\s+java\s+/ // match "java" with one or more spaces before and after

- **The RegExp Object**
  - Regular expressions are represented as RegExp objects
  - In addition to RegExp() constructor, RegExp object supports three methods and some properties

JavaScript Pattern Matching with Regular Expressions

- **The RegExp Object**
  - The RegExp() constructor takes one or **two string arguments** and c**reates** a **new RegExp object**
    - The **first argument** is a string that contains the body of the regular expression (text that appears within slashes of a regular expression literal)
    - The **second argument** is **optional**. If used, it means the regular expression flags. The flags should be *g*, *i*, *m* or a combination of these letters (*g* means global (search all), *i* means case insensitive (both upper and lower case allowed) and *m* means multiline mode pattern matching)
  - **RegExp Methods**
    - There are two methods
    - The methods are exec() and test()

JavaScript Pattern Matching with Regular Expressions

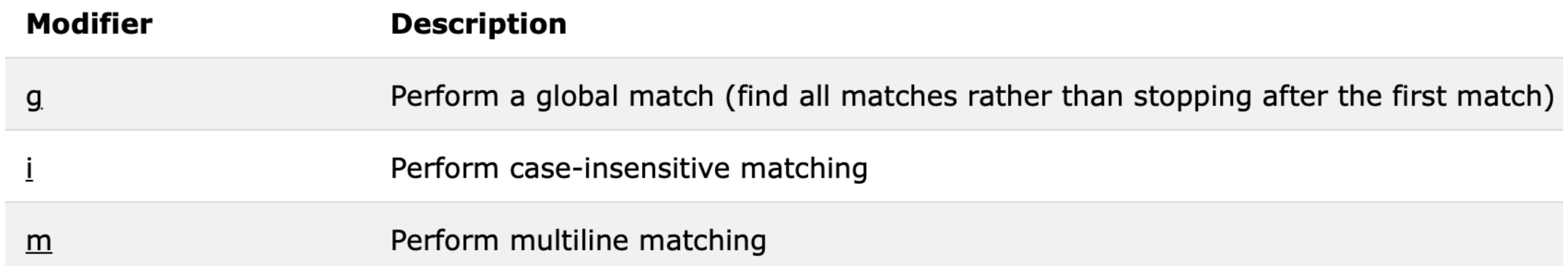
- **The RegExp Object**
  - The exec() method execute on the string to match the pattern
  - The test() method takes a string argument and returns true if the string is in the pattern else false
  - Example code1:

```
var pattern =/Java/g;
var text="JavaScript is more fun than Java!";
var result;
while(result=pattern.exec(text)) !=null) {
    alert("Matched'" + result[0] + "'" +  "at position" + result.index + "; next
search begins at  " + pattern.lastindex);
}
```

JavaScript Pattern Matching with Regular Expressions

- **The RegExp Methods**
  - Example code2:

```
var pattern =/java/i;
pattern.test(“JavaScript”); // This returns true
```



Modifier	Description
g	Perform a global match (find all matches rather than stopping after the first match)
i	Perform case-insensitive matching
m	Perform multiline matching

Queen Mary
University of London

JavaScript Pattern Matching with Regular Expressions

More examples:

```
/ab+c/i;
new RegExp(/ab+c/, 'i'); // literal notation
```

Or

```
new RegExp('ab+c', 'i'); // constructor
```

```
var re = /\w+/;
```

Or

```
var re = new RegExp('\\w+');
```

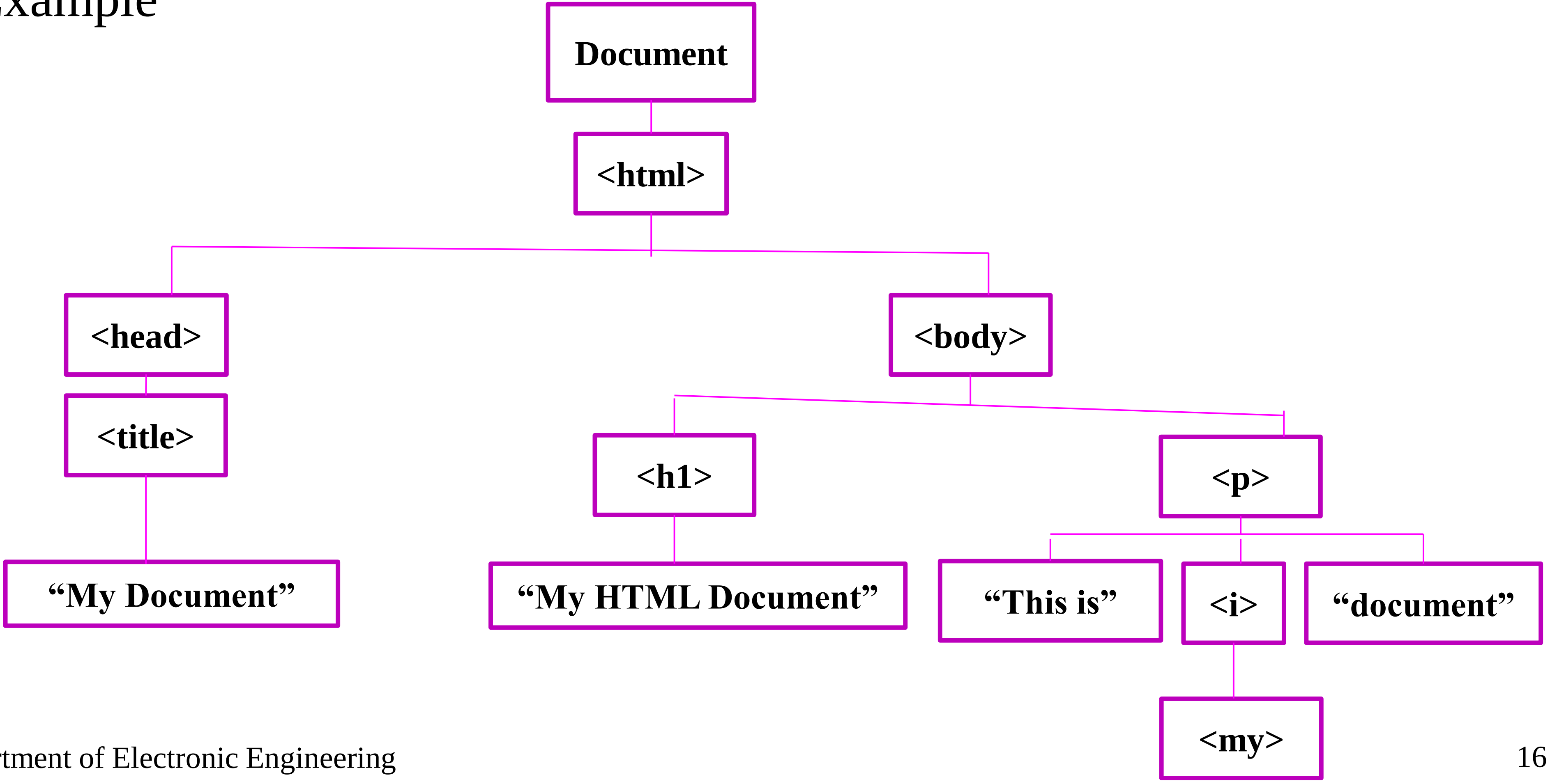
Scripting Documents

- Revision: DOM-Document Object Model
- Example

```
<html>
   <head>
        <title> My Document</title>
    </head>
    <body>
        <h1> My HTML Document</h1>
        <p> This is <i>my</i>document.
</html>
```

Scripting Documents

- Revision: DOM-Document Object Model
- Example



Scripting Documents

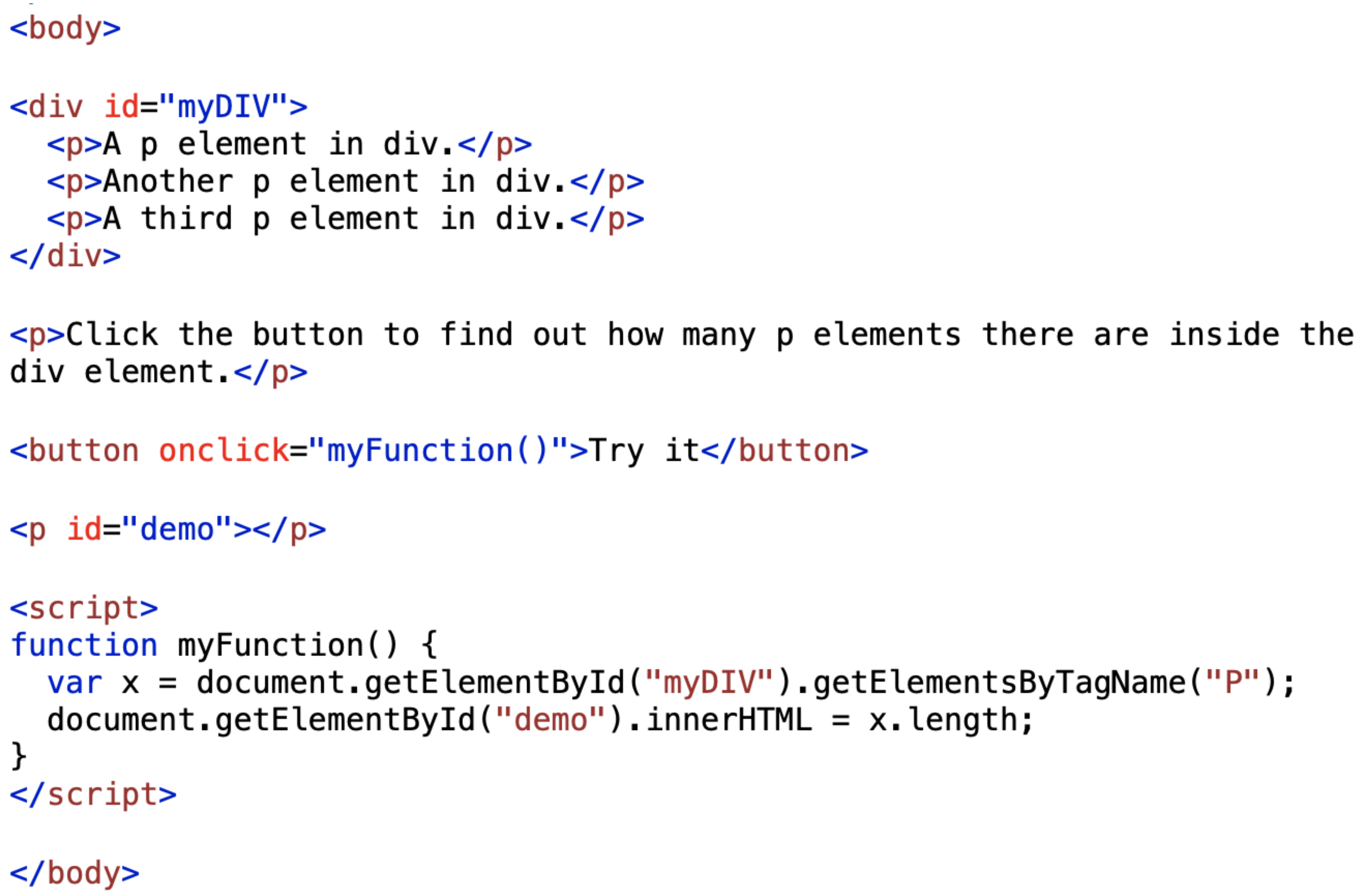
- **Selecting Document Elements**
  - Client-side data works on manipulating documents
  - To manipulate documents, you need to obtain or select the elements of the documents
  - You can **query** a document using a specified ID attribute, specified name attribute, tag name, etc
- **Selecting documents by ID**
  - The HTML element can have an “id” attribute
  - The value of this attribute must be unique within the document
  - You can use the getElementById() method to select an element based on this ID
  - E.g. var selectme=document.getElementById(“myid”);

Scripting Documents

- **Selecting documents by Name**
  - You can use the HTML element name attribute to select items within the documents
  - The name attribute does not have to be unique
  - Multiple elements may have the same, eg in radio buttons
  - Use the getElementByName() method to select items
  - E.g. var radiobuttons=document.getElementByName(“colour_types”);
- **Selecting documents by Tag**
  - getElememntsByTagName() Method

Scripting Documents

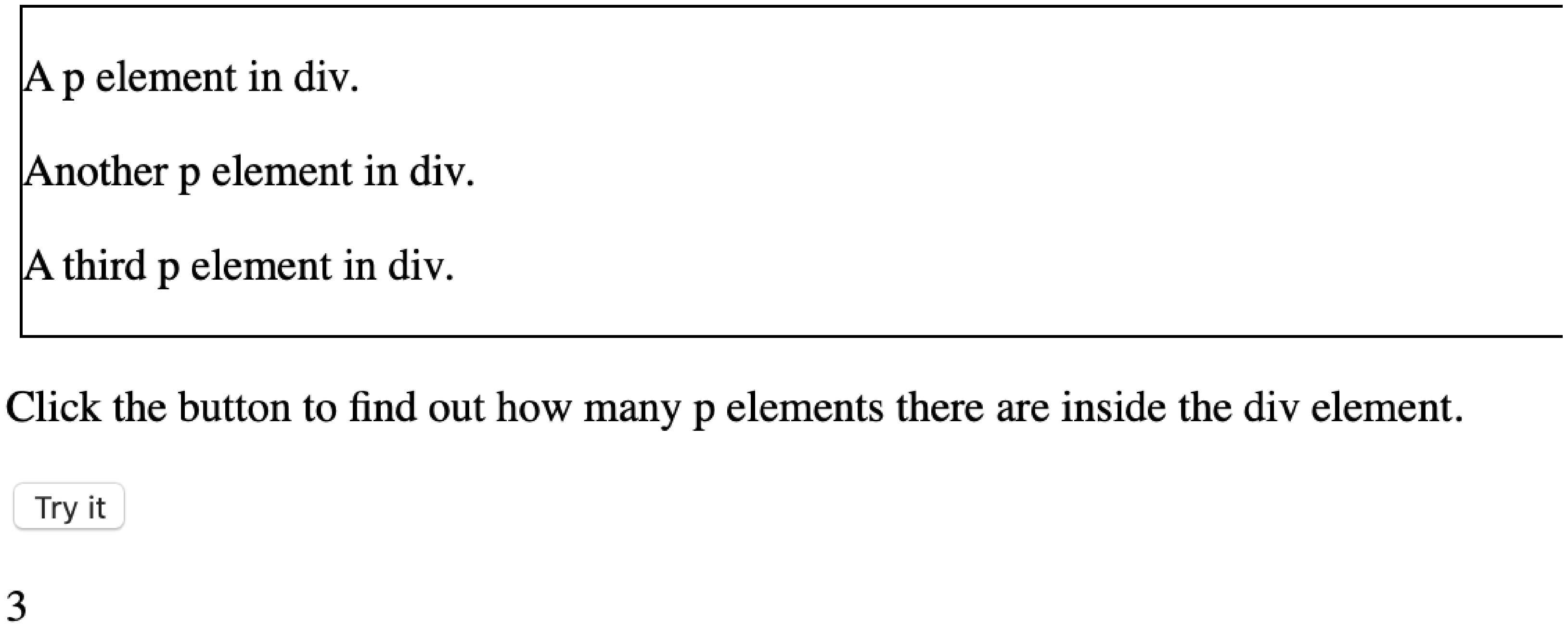
Example:



```
<body>

<div id="myDIV">
  <p>A p element in div.</p>
  <p>Another p element in div.</p>
  <p>A third p element in div.</p>
</div>

<p>Click the button to find out how many p elements there are inside the
div element.</p>

<button onclick="myFunction()">Try it</button>

<p id="demo"></p>

<script>
function myFunction() {
  var x = document.getElementById("myDIV").getElementsByTagName("P");
  document.getElementById("demo").innerHTML = x.length;
}
</script>

</body>
```

Scripting Documents

Example output:



A p element in div.

Another p element in div.

A third p element in div.

Click the button to find out how many p elements there are inside the div element.

3

Source: https://www.w3schools.com/jsref/met_element_getelementsbytagname.asp