JAVASCRIPT

目錄

Overview

改變HTML內容

What Can JavaScript Do?

JavaScript can change HTML content.

Click Me!

What Can JavaScript Do?

Hello Kun Shan University!

Click Me!

ex02-01.html

改變HTML內容

```
<!DOCTYPE html>
<html>
<body>
  <h2>What Can JavaScript Do?</h2>
  <p id="demo">JavaScript can change HTML content.</p>
  <button type="button" onclick='document.getElementById("demo").innerHTML
    = "Hello Kun Shan University!"'>Click Me!</button>
</body>
</html>
```

- One of many JavaScript HTML methods is getElementById().
- The example below "finds" an HTML element (with id="demo"), and changes the element content (innerHTML) to "Hello KUN Shan University":

```html
<!DOCTYPE html>
<html>
<body>
  <h2>What Can JavaScript Do?</h2>
  <p id="demo">JavaScript can change HTML content.</p>
  <button type="button" onclick='document.getElementById("demo").innerHTML
     = "Hello Kun Shan University!"'>Click Me!</button>
</body>
</html>
```

ex02-01.html

JavaScript accepts both double and single quotes:

```html
<!DOCTYPE html>
<html>
<body>
  <h2>What Can JavaScript Do?</h2>
  <p id="demo">JavaScript can change HTML content.</p>
  <button type="button" onclick="document.getElementById('demo').innerHTML
     = 'Hello Kun Shan University!'">Click Me!</button>
</body>
</html>
```

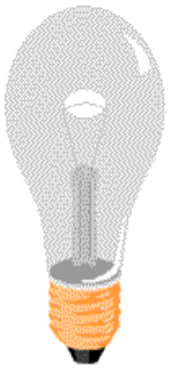
ex02-01a.html

改變Attribute Value

What Can JavaScript Do?

JavaScript can change HTML attribute values.

In this case JavaScript changes the value of the src (source) attribute of an image.

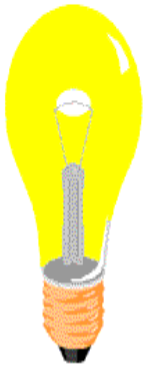


Turn on the light Turn off the light

What Can JavaScript Do?

JavaScript can change HTML attribute values.

In this case JavaScript changes the value of the src (source) attribute of an image.



Turn on the light Turn off the light

ex02-02.html

改變Attribute Value

- <button onclick=

```
                    "document.getElementById('myImage').src =
                        'pic_bulbon.gif'">
                        Turn on the light
</button>
<img id="myImage“
        src="pic_bulboff.gif“
        style="width:100px">
```

```html
<!DOCTYPE html>
<html>
<body>
  <h2>What Can JavaScript Do?</h2>
  <p>JavaScript can change HTML attribute values.</p>
  <p>In this case JavaScript changes the value of
     the src (source) attribute of an image.</p>
  <button onclick
          ="document.getElementById('myImage').src='pic_bulbon.gif'">
          Turn on the light</button>
  <img id="myImage" src="pic_bulboff.gif" style="width:100px">
  <button onclick
          ="document.getElementById('myImage').src='pic_bulboff.gif'">
          Turn off the light</button>
</body>
</html>
```

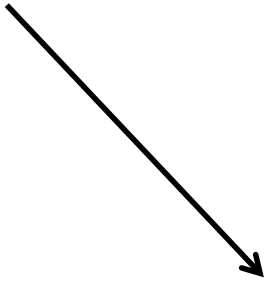
ex02-02.html

改變CSS

What Can JavaScript Do?

JavaScript can change the style of an HTML element.

Click Me!



What Can JavaScript Do?

JavaScript can change the style of an HTML element.

Click Me!

ex02-03.html

改變CSS

```
<p id="demo">
        JavaScript can change the style of an HTML element.
</p>
<button type="button"
        onclick=
            "document.getElementById('demo').style.fontSize='35px'">
                    Click Me!
</button>
```

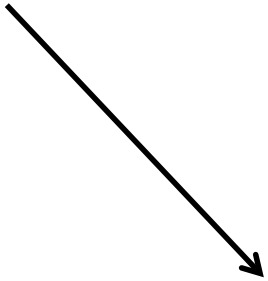
```html
<!DOCTYPE html>
<html>
<body>

 <h2>What Can JavaScript Do?</h2>
 <p id="demo">JavaScript can change the style of an HTML element.</p>
 <button type="button"
         onclick=
         "document.getElementById('demo').style.fontSize='35px'">
         Click Me!</button>

</body>
</html>
```

隱藏HTML元素

What Can JavaScript Do?

JavaScript can hide HTML elements.

Click Me!



What Can JavaScript Do?

Click Me!

ex02-04.html

隱藏HTML元素

- JavaScript can hide HTML elements.

  <button type="button"
  onclick=
  "document.getElementById('demo').style.display='none'">
  Click Me!
  </button>

```html
<!DOCTYPE html>
<html>
<body>

 <h2>What Can JavaScript Do?</h2>
 <p id="demo">JavaScript can hide HTML elements.</p>
 <button type="button"
         onclick=
         "document.getElementById('demo').style.display='none'">
         Click Me!</button>

</body>
</html>
```

JavaScript Structure

Used <script> Tag

- In HTML, JavaScript code is inserted between <script> and </script> tags.

JavaScript in Body

My First JavaScript
Kun Shan University

ex02-05.html

- Old JavaScript examples may use a type attribute: <script type="text/javascript">. The type attribute is not required. JavaScript is the default scripting language in HTML.

```
<!DOCTYPE html>
<html>
<body>
 <h2>JavaScript in Body</h2>
 <p id="demo"></p>
 <script>
   document.getElementById("demo").innerHTML
    = "My First JavaScript"+"<br>"+"Kun Shan University";
 </script>
</body>
</html>
```

比較

無Script標籤

```
<!DOCTYPE html>
<html>
<body>
  <h2>What Can JavaScript Do?</h2>
  <p id="demo">JavaScript can change HTML content.</p>
  <button type="button" onclick="document.getElementById('demo').innerHTML
     = 'Hello Kun Shan University!'">Click Me!</button>
</body>
</html>
```

有Script標籤

```
<!DOCTYPE html>
<html>
<body>
 <h2>JavaScript in Body</h2>
 <p id="demo"></p>
 <script>
   document.getElementById("demo").innerHTML
     = "My First JavaScript"+"<br>"+"Kun Shan University";
 </script>
</body>
</html>
```

JavaScript Functions and Events

- A JavaScript function is a block of JavaScript code, that can be executed when "called" for. For example, a function can be called when an **event** occurs, like when the user clicks a button.
- You can place any number of scripts in an HTML document.
- Scripts can be placed in the <body>, or in the <head> section of an HTML page, or in both.

JavaScript in <head>

JavaScript in Head

A Paragraph.

Try it

JavaScript in Head

Paragraph changed.

Try it

ex02-06.html

```html
<!DOCTYPE html>
<html>
 <head>
   <script>
     function myFunction() {
        document.getElementById("demo").innerHTML
        = "Paragraph changed.";
     }
   </script>
 </head>
 <body>
  <h2>JavaScript in Head</h2>
  <p id="demo">A Paragraph.</p>
  <button type="button" onclick="myFunction()">
     Try it
  </button>
 </body>
</html>
```

Exercise-改成有Script標籤

```
<!DOCTYPE html>
<html>
<body>
  <h2>What Can JavaScript Do?</h2>
  <p id="demo">JavaScript can change HTML content.</p>
  <button type="button" onclick="document.getElementById('demo').innerHTML
     = 'Hello Kun Shan University!'">Click Me!</button>
</body>
</html>
```

ex02-01a.html

What Can JavaScript Do?

JavaScript can change HTML content.

Click Me!

→

What Can JavaScript Do?

Hello Kun Shan University!

Click Me!

ex02-01b.html

JavaScript in <body>

JavaScript in Body

A Paragraph.

Try it

JavaScript in Body

Paragraph changed.

Try it

ex02-07.html

- Placing scripts at the bottom of the <body> element improves the display speed, because script interpretation slows down the display.

```
<!DOCTYPE html>
<html>
  <body>
    <h2>JavaScript in Head</h2>
    <p id="demo">A Paragraph.</p>
    <button type="button" onclick="myFunction()">
      Try it
    </button>
    <script>
      function myFunction() {
        document.getElementById("demo").innerHTML
        = "Paragraph changed.";
      }
    </script>
  </body>
</html>
```

External JavaScript

External JavaScript

- Scripts can also be placed in external files.
- External scripts are practical when the same code is used in many different web pages.
- JavaScript files have the file extension **.js**. To use an external script, put the name of the script file in the src (source) attribute of a <script> tag:

External JavaScript

External JavaScript

A Paragraph.

Try it

(myFunction is stored in an external file called "ex02-08j.js")

External JavaScript

Paragraph changed.

Try it

(myFunction is stored in an external file called "ex02-08j.js")

ex02-08.html

External scripts cannot contain <script> tags.

```html
<!DOCTYPE html>
<html>
 <body>
   <h2>External JavaScript</h2>
   <p id="demo">A Paragraph.</p>
   <button type="button" onclick="myFunction()">Try it</button>
   <p>(myFunction is stored in
       an external file called "ex02-08j.js")</p>
   <script src="ex02-08js.js"> </script>

 </body>
</html>
```

ex02-08.html

```javascript
function myFunction() {
  document.getElementById("demo").innerHTML
    = "Paragraph changed.";
}
```

ex02-08js.html

External JavaScript Advantages

- Placing scripts in external files has some advantages:
  - It separates HTML and code
  - It makes HTML and JavaScript easier to read and maintain
  - Cached JavaScript files can speed up page loads
- To add several script files to one page - use several script tags:

```
<script src="myScript1.js"></script>
<script src="myScript2.js"></script>
```

External References

- External scripts can be referenced with a full URL or with a path relative to the current web page.
- This example uses a full URL to link to a script:

Example

```
<script src="https://www.w3schools.com/js/myScript1.js"></script>
```

- This example uses a script located in a specified folder on the current web site:

Example

```
<script src="/js/myScript1.js"></script>
```

- This example links to a script located in the same folder as the current page:

Example

```
<script src="myScript1.js"></script>
```

Output

JavaScript Display Possibilities

- JavaScript can "display" data in different ways:
  - Writing into an HTML element, using innerHTML.(寫入到 HTML 元素)
  - Writing into the HTML output using document.write().(寫到 HTML 文件中)
  - Writing into an alert box, using window.alert().(彈出警告框)
  - Writing into the browser console, using console.log().(寫入到瀏覽器的控制檯)
- 如果您的瀏覽器支援除錯，你可以使用 console.log() 方法在瀏覽器中顯示 JavaScript 值。瀏覽器中使用 F12 來啟用除錯模式， 在除錯視窗中點選 “Console” 選單。

Using innerHTML

- To access an HTML element, JavaScript can use the document.getElementById(id) method.
- The id attribute defines the HTML element. The innerHTML property defines the HTML content:

My First Web Page

My First Paragraph.

11

ex02-09.html

```html
<!DOCTYPE html>
<html>
<body>

  <h2>My First Web Page</h2>
  <p>My First Paragraph.</p>
  <p id="demo"></p>
  <script>
    document.getElementById("demo").innerHTML
     = 5 + 6;
  </script>

</body>
</html>
```

My First Web Page

My First Paragraph.

11

Using document.write()

- For testing purposes, it is convenient to use document.write():
- Using document.write() after an HTML document is loaded, will **delete all existing HTML**

My First Web Page

My first paragraph.

Never call document.write after the document has finished loading. It will overwrite the whole document.

11

ex02-10.html

Using document.write()

```html
<!DOCTYPE html>
<html>
<body>

<h2>My First Web Page</h2>
<p>My first paragraph.</p>

<p>Never call document.write after the document has finished loading.
It will overwrite the whole document.</p>

<script>
document.write(5 + 6);
</script>

</body>
</html>
```

My First Web Page

My first paragraph.

Never call document.write after the document has finished loading. It will overwrite the whole document.

11

Syntax & Variables

JavaScript Syntax

- JavaScript syntax is the set of rules, how JavaScript programs are constructed:

```
var x, y, z;        // Declare Variables
x = 5; y = 6;       // Assign Values
z = x + y;          // Compute Values
```

- The JavaScript syntax defines two types of values:
  - Fixed values
  - Variable values
- Fixed values are called **Literals (非變數設定的值)**. Variable values are called **Variables(變數設定的值)**.

JavaScript Literals

- The two most important syntax rules for fixed values are:
  - 1. **Numbers** are written with or without decimals:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Numbers</h2>

<p>Number can be written with or without decimals.</p>

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

<script>
document.getElementById("demo").innerHTML = 10.50;
</script>

</body>
</html>
```

JavaScript Numbers

Number can be written with or without decimals.

10.5

  - 2. **Strings** are text, written within double or single quotes:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Strings</h2>

<p>Strings can be written with double or single quotes.</p>

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

<script>
document.getElementById("demo").innerHTML = 'John Doe';
</script>

</body>
</html>
```

JavaScript Strings

Strings can be written with double or single quotes.

John Doe

JavaScript Variables

- JavaScript variables are containers for storing data values.

```html
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Variables</h2>

<p>In this example, x, y, and z are variables.</p>

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

<script>
 var x = 5;
 var y = 6;
 var z = x + y;
 document.getElementById("demo").innerHTML =
 "The value of z is: " + z;
</script>

</body>
</html>
```

JavaScript Variables

In this example, x, y, and z are variables.

The value of z is: 11

JavaScript Identifiers

- All JavaScript **variables** must be **identified** with **unique names**. These unique names are called **identifiers**. The general rules for constructing names for variables (unique identifiers) are:
  - Names can contain letters, digits, underscores, and dollar signs.
  - Names must begin with a letter
  - Names can also begin with $ and _ (but we will not use it in this tutorial)
  - Names are case sensitive (y and Y are different variables)
  - Reserved words (like JavaScript keywords) cannot be used as names
- JavaScript identifiers are case-sensitive.

JavaScript Data Types

- JavaScript variables can hold numbers like 100 and text values like "John Doe". In programming, text values are called text strings.
- JavaScript can handle many types of data, but for now, just think of numbers and strings.
- Strings are written inside double or single quotes. Numbers are written without quotes. If you put a number in quotes, it will be treated as a text string.

```
var pi = 3.14;
var person = "John Doe";
var answer = 'Yes I am!';
```

- You can also add strings, but strings will be concatenated:

```
var x = "John" + " " + "Doe";
```

JavaScript Arithmetic

- As with algebra, you can do arithmetic with JavaScript variables, using operators like = and +:

```
var x = 5 + 2 + 3;
```

JavaScript Variables

The result of adding 5 + 2 + 3:

10

```
var x = "5" + 2 + 3;
```

JavaScript Variables

The result of adding "5" + 2 + 3:

523

```
var x = 2 + 3 + "5";
```

JavaScript Variables

The result of adding 2 + 3 + "5":

55

JavaScript Comparison Operators

Operator	Description
==	equal to
===	equal value and equal type
!=	not equal
!==	not equal value or not equal type
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
?	ternary operator

Data Types

JavaScript Data Types

- JavaScript variables can hold many **data types**: numbers, strings, objects and more:

```
var length = 16;                                  // Number
var lastName = "Johnson";                         // String
var x = {firstName:"John", lastName:"Doe"};       // Object
```

JavaScript Types are Dynamic

- JavaScript has dynamic types. This means that the same variable can be used to hold different data types:

```
var x;              // Now x is undefined
x = 5;              // Now x is a Number
x = "John";         // Now x is a String
```

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Data Types</h2>

<p>JavaScript has dynamic types.
This means that the same variable
can be used to hold different data
types:</p>
<p id="demo"></p>
<script>
  var x;          // Now x is
undefined
  x = 5;          // Now x is a
Number
  x = "John";       // Now x is a
String

 document.getElementById("demo").in
nerHTML = x;
</script>
</body>
</html>
```

JavaScript Data Types

JavaScript has dynamic types. This means that the same variable can be used to hold different data types:

John

JavaScript Arrays

- JavaScript arrays are written with square brackets. Array items are separated by commas. The following code declares (creates) an array called cars, containing three items (car names):
- Array indexes are zero-based, which means the first item is [0], second is [1], and so on.

```
var cars = ["Saab", "Volvo", "BMW"];
```

JavaScript Objects

- JavaScript objects are written with curly braces { }. Object properties are written as name:value pairs, separated by commas.

```
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
```

The typeof Operator

- You can use the JavaScript typeof operator to find the type of a JavaScript variable. The typeof operator returns the type of a variable or an expression:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript typeof</h2>
<p>The typeof operator returns the
type of a variable or an
expression.</p>

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

<script>
document.getElementById("demo").inn
erHTML =
typeof "" + "<br>" +
typeof "John" + "<br>" +
typeof "John Doe";
</script>
</body>
</html>
```

JavaScript typeof

The typeof operator returns the type of a variable or an expression.

string
string
string

The typeof Operator

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript typeof</h2>
<p>The typeof operator returns the
type of a variable or an expression.
</p>

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

<script>
document.getElementById("demo").innerH
TML =
typeof 0 + "<br>" +
typeof 314 + "<br>" +
typeof 3.14 + "<br>" +
typeof (3) + "<br>" +
typeof (3 + 4);
</script>

</body>
</html>
```

JavaScript typeof

The typeof operator returns the type of a variable or an expression.

number
number
number
number
number

Empty Values

```
var car = "";    // The value is "", the typeof is "string"
```

- In JavaScript null is “nothing”. It is supposed to be something that doesn‘t exist. Unfortunately, in JavaScript, the data type of null is an object.
- You can consider it a bug in JavaScript that typeof null is an object. It should be null. You can empty an object by setting it to null:

```
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
person = null;    // Now value is null, but type is still an object
```

Difference Between Undefined and Null

- Difference Between Undefined and Null

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript</h2>

<p>Undefined and null are equal in value but
different in type:</p>

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

<script>
document.getElementById("demo").innerHTML =
typeof undefined + "<br>" +
typeof null + "<br><br>" +
(null === undefined) + "<br>" +
(null == undefined);
</script>

</body>
</html>
```

JavaScript

Undefined and null are equal in value but different in type:

undefined
object

false
true

Primitive Data

- A primitive data value is a single simple data value with no additional properties and methods. The typeof operator can return one of these primitive types:

  - `string`
  - `number`
  - `boolean`
  - `undefined`

```
typeof "John"                // Returns "string"
typeof 3.14                  // Returns "number"
typeof true                  // Returns "boolean"
typeof false                 // Returns "boolean"
typeof x                     // Returns "undefined" (if x has no value)
```

Complex Data

- The typeof operator can return one of two complex types:
  - `function`
  - `object`
- The typeof operator returns “object” for objects, arrays, and null. The typeof operator does not return "object" for functions.

```
typeof {name:'John', age:34} // Returns "object"
typeof [1,2,3,4]             // Returns "object" (not "array", see note below)
typeof null                  // Returns "object"
typeof function myFunc(){}   // Returns "function"
```

p.s. The typeof operator returns "object" for arrays because in JavaScript arrays are objects.

Function

JavaScript Functions

- A JavaScript function is a block of code designed to perform a particular task. A JavaScript function is executed when "something" invokes it (calls it).

```
<!DOCTYPE html>
<html>
 <body>

  <h2>JavaScript Functions</h2>
  <p>This example calls a function which
performs a
     calculation, and returns the result:
</p>
  <p id="demo"></p>

  <script>
   function myFunction(p1, p2) {
     return p1 * p2;
   }
  document.getElementById("demo").innerHTML
=
  myFunction(4, 3);
  </script>
 </body>
</html>
```

JavaScript Functions

This example calls a function which performs a calculation, and returns the result:

12

JavaScript Function Syntax

- A JavaScript function is defined with the function keyword, followed by a **name**, followed by parentheses **()**. Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).
- The parentheses may include parameter names separated by commas: **(*parameter1, parameter2, ...*)** The code to be executed, by the function, is placed inside curly brackets: **{ }**

```
function name(parameter1, parameter2, parameter3) {
  // code to be executed
}
```

Function Return

- Calculate the product of two numbers, and return the result:

```
var x = myFunction(4, 3);   // Function is called, return value will end up in x

function myFunction(a, b) {
  return a * b;             // Function returns the product of a and b
}
```

The result in x will be:

```
12
```

Local Variables

- Variables declared within a JavaScript function, become **LOCAL** to the function.
- Local variables can only be accessed from within the function.

```
// code here can NOT use carName

function myFunction() {
  var carName = "Volvo";
  // code here CAN use carName
}

// code here can NOT use carName
```

Extend

```html
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Functions</h2>

<p>This example calls a function which performs a
calculation and returns the result:</p>

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

<script>
  var x = myFunction(4, 3);
  var y = 5;
  document.getElementById("demo").innerHTML = x;

  function myFunction(a, b) {
    return a * b * y ;
  }
</script>

</body>
</html>
```

JavaScript Functions

This example calls a function which performs a calculation and returns the result:

NaN

```html
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Functions</h2>

<p>This example calls a function which performs a
calculation and returns the result:</p>

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

<script>
  var x = myFunction(4, 3);
  var y = 5;

  function myFunction(a, b) {
    return a * b  ;
  }

   document.getElementById("demo").innerHTML
   = x * y;

</script>

</body>
</html>
```

JavaScript Functions

This example calls a function which performs a calculation and returns the result:

60

Objects

JavaScript Objects

- In real life, a car is an **object**. A car has **properties** like weight and color, and **methods** like start and stop:

Object	Properties	Methods
	car.name = Fiat	car.start()
	car.model = 500	car.drive()
	car.weight = 850kg	car.brake()
	car.color = white	car.stop()

- All cars have the same **properties**, but the property **values** differ from car to car. All cars have the same **methods**, but the methods are performed **at different times**.

JavaScript Objects

- Objects are variables too. But objects can contain many values.
- The values are written as **name:value** pairs (name and value separated by a colon).

```
var car = {type:"Fiat", model:"500", color:"white"};
```

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Objects</h2>

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

<script>
// Create an object:
var car = {type:"Fiat", model:"500",
color:"white"};

// Display some data from the object:
document.getElementById("demo").innerHT
ML = "The car type is " + car.type;
</script>

</body>
</html>
```

JavaScript Objects

The car type is Fiat

Objects Definition

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Objects</h2>

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

<script>
// Create an object:
var person = {
  firstName: "John",
  lastName: "Doe",
  age: 50,
  eyeColor: "blue"
};

// Display some data from the object:
document.getElementById("demo").innerHTML =
person.firstName + " is " + person.age + "
years old.";
</script>

</body>
</html>
```

JavaScript Objects

John is 50 years old.

Object Properties

- The **name:values** pairs in JavaScript objects are called **properties**:

Property	Property Value
firstName	John
lastName	Doe
age	50
eyeColor	blue

Accessing Object Properties

- You can access object properties in two ways:

```
objectName.propertyName
```

or

```
objectName["propertyName"]
```

Object Methods

- Objects can also have **methods**. Methods are **actions** that can be performed on objects. Methods are stored in properties as **function definitions**. A method is a function stored as a property.

Property	Property Value
firstName	John
lastName	Doe
age	50
eyeColor	blue
fullName	function() {return this.firstName + " " + this.lastName;}

```
var person = {
  firstName: "John",
  lastName : "Doe",
  id       : 5566,
  fullName : function() {
    return this.firstName + " " + this.lastName;
  }
};
```

The **this** Keyword

- In a function definition, this refers to the "owner" of the function.
- In the example above, this is the **person object** that "owns" the fullName function.
- In other words, `this.firstName` means the `firstName` property of **this object**.

Accessing Object Methods

- You access an object method with the following syntax:

```
objectName.methodName()
```

Example

```
name = person.fullName();
```

Do Not Declare Strings, Numbers, and Booleans as Objects!

When a JavaScript variable is declared with the keyword "`new`", the variable is created as an object:

```
var x = new String();        // Declares x as a String object
var y = new Number();        // Declares y as a Number object
var z = new Boolean();       // Declares z as a Boolean object
```

Avoid `String`, `Number`, and `Boolean` objects. They complicate your code and slow down execution speed.

Arrays

JavaScript Arrays

- JavaScript arrays are used to store multiple values in a single variable.

```
var cars = ["Saab", "Volvo", "BMW"];
```

What is an Array?

- An array is a special variable, which can hold more than one value at a time.
- If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

```
var car1 = "Saab";
var car2 = "Volvo";
var car3 = "BMW";
```

- However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300? The solution is an array!
- An array can hold many values under a single name, and you can access the values by referring to an index number.

Creating an Array

Syntax:

```
var array_name = [item1, item2, ...];
```

Example

```
var cars = ["Saab", "Volvo", "BMW"];
```

Creating an Array

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Arrays</h2>

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

 <script>
  var cars = [
   "Saab",
    "Volvo",
  "BMW"
 ];
 document.getElementById("demo").innerHTML =
 cars;
</script>

</body>
</html>
```

JavaScript Arrays

Saab,Volvo,BMW

Using the JavaScript Keyword new

- The following example also creates an Array, and assigns values to it:

```
var cars = new Array("Saab", "Volvo", "BMW");
```

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Arrays</h2>

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

<script>
var cars = new Array("Saab", "Volvo",
"BMW");
document.getElementById("demo").innerHTML =
cars;
</script>

</body>
</html>
```

JavaScript Arrays

Saab,Volvo,BMW

Access the Full Array

With JavaScript, the full array can be accessed by referring to the array name:

Example

```
var cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars;
```

- The two examples above do exactly the same. There is no need to use new Array().
  For simplicity, readability and execution speed, use the first one (the array literal method).

Access the Elements of an Array

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Arrays</h2>

<p>JavaScript array elements are accessed
using numeric indexes (starting from 0).
</p>

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

<script>
var cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML
= cars[0];
</script>

</body>
</html>
```

JavaScript Arrays

JavaScript array elements are accessed using numeric indexes (starting from 0).

Saab

Arrays are Objects

- Arrays are a special type of objects. The typeof operator in JavaScript returns "object" for arrays.
- But, JavaScript arrays are best described as arrays. Arrays use **numbers** to access its "elements". In this example, person[0] returns John:

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

<script>
var person = ["John", "Doe", 46];
document.getElementById("demo").innerHTML =
person[0];
</script>
```

John

Array Elements Can Be Objects

- JavaScript variables can be objects. Arrays are special kinds of objects. Because of this, you can have variables of different types in the same Array. You can have objects in an Array. You can have functions in an Array. You can have arrays in an Array:

```
myArray[0] = Date.now;
myArray[1] = myFunction;
myArray[2] = myCars;
```

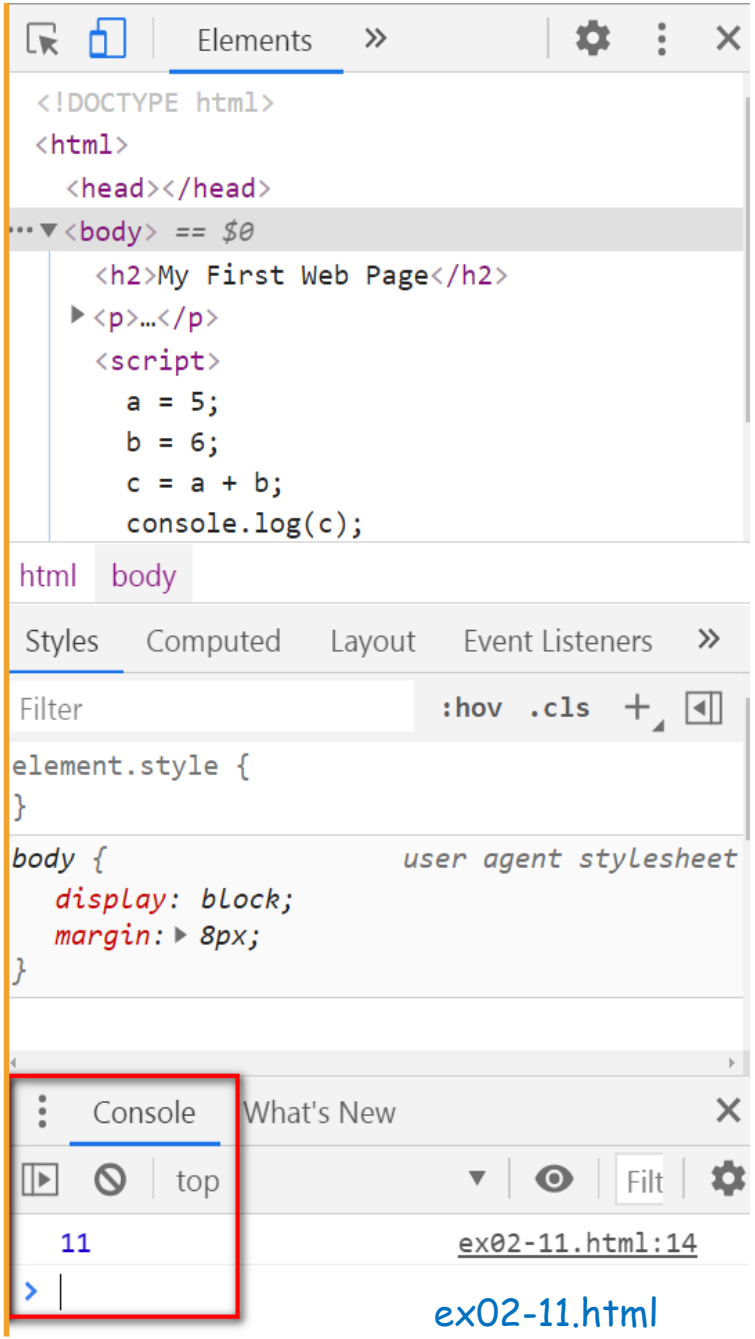
Debugging

JavaScript Debugging

- Errors can (will) happen, every time you write some new computer code.
- Debugging is not easy. But fortunately, all modern browsers have a built-in JavaScript debugger.
- Built-in debuggers can be turned on and off, forcing errors to be reported to the user.
- With a debugger, you can also set breakpoints (places where code execution can be stopped), and examine variables while the code is executing.
- Normally, otherwise follow the steps at the bottom of this page, you activate debugging in your browser with the F12 key, and select "Console" in the debugger menu.

1.The console.log()

```
<!DOCTYPE html>
<html>
<body>

<h2>My First Web Page</h2>
<p>Activate debugging in your browser
   (Chrome, IE, Firefox) with F12,
    and select "Console"
    in the debugger menu.</p>
 <script>
   a = 5;
   b = 6;
   c = a + b;
   console.log(c);
 </script>
</body>
</html>
```



ex02-11.html

2. Setting Breakpoints

- In the debugger window, you can set breakpoints in the JavaScript code. At each breakpoint, JavaScript will stop executing, and let you examine JavaScript values. After examining values, you can resume the execution of code (typically with a play button).
- The debugger keyword stops the execution of JavaScript, and calls (if available) the debugging function.

```
<!DOCTYPE html>
<html>
<head>
</head>

<body>

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

<p>With the debugger turned on, the code
below should stop executing before it
executes the third line.</p>

<script>
var x = 15 * 5;
debugger;
document.getElementById("demo").innerHTML =
x;
</script>

</body>
</html>
```

75

With the debugger turned on, the code below should stop executing before it executes the third line.

Example

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
  <p id="demo1"> </p>
  <p id="demo2"> </p>
  <p>With the debugger turned on, the
     code below
     should stop executing before it
     executes the third line.</p>
     <script>
     var x = 15 * 5;
     document.getElementById("demo1").innerHTML = x;
     debugger;
     var y= 10 + 2;
     document.getElementById("demo2").innerHTML = y;
     </script>
</body>
</html>
```

ex02-12.html

In Firefox

- 由於程式碼是一行一行執行，在程式碼中放入 debugger 語法，便可以在程式碼執行到該行數時觸發除錯器，並在除錯器中檢查你要除錯的內容。

除錯器 網路 樣式編輯器 效能 記憶體 儲存空間 輔助功能 應用程式

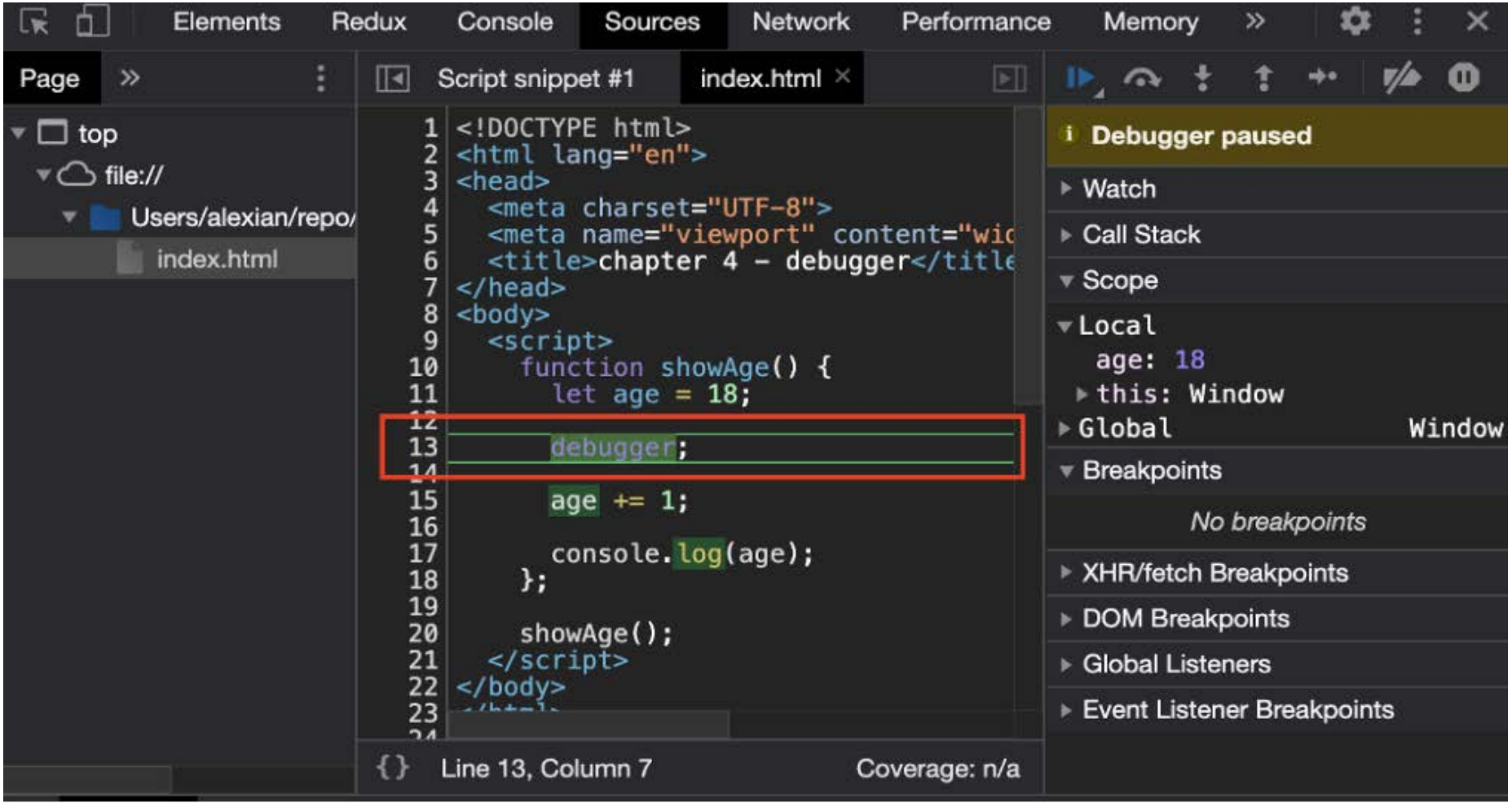
大綱 ex02-12.html ×

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4  </head>
5
6  <body>
7
8   <p id="demo1"> </p>
9   <p id="demo2"> </p>
10  <p>With the debugger turned on, the code below
11     should stop executing before it executes the third line.</p>
12
13   <script>
14   var x = 15 * 5;
15   docu  undefined  mentById("demo1").innerHTML = x;
16     de
17   var y= 10 + 2;
18   document.getElementById("demo2").innerHTML = y;
19   </script>
20
21 </body>
22 </html>
```

ex02-12.html

Example

- 可以看到現在執行到第 13 行，也就是 debugger 語法時停止了，這個狀態稱為 breakpoint ，在這個狀態下，你可以在 console 中取得該 breakpoint 前執行的內容。例如該斷點下 age 的值仍為 18



Extra Talks

什麼是箭頭函數（Arrow function）

- 我們撰寫函數的方法：

```
function greeting(){
  console.log("Welcome to PJCHENder!");
}

greeting(); // "Welcome to PJCHENder!"
```

- 們可以把它改成箭頭函數的寫法，它會變成下面這樣：

```
var greeting = () => {
    console.log("Welcome to PJCHENder!")

}
greeting(); // "Welcome to PJCHENder!"
```

- 沒有參數的時候要記得加上空括號.如果只是要回傳某個值，可以省略 return

```
var greeting = () => "Welcome to PJCHENder!";
console.log(greeting()); // "Welcome to PJCHENder!"

// 上面的箭頭函數等同於下面原本的寫法
var greeting = function(){
    return "Welcome to PJCHENder!";
}
```

箭頭函數帶入參數值

- 當我們的函式擁有兩個 參數時，我們一樣要使用括號來帶入參數，寫法像是下面這樣子：

```
var add = (a,b) => a+b;
console.log(add(3,5));   //  8

// 上面的箭頭函數等同於下面原本的寫法
var add = function(a, b){
    return a+b;
}
```

- 當函數只有一個參數時，不需要使用括號

```
var greeting = person => "Hello, "+ person;
console.log(greeting("pjchender")); //  "Hello, pjchender"

// 上面的箭頭函數等同於下面原本的寫法
var greeting = function(person){
    return "Hello, "+ person;
}
```

The End