

CT053-3-1

Fundamentals of Web Design & Development



A · P · U
ASIA PACIFIC UNIVERSITY
OF TECHNOLOGY & INNOVATION

JavaScript – Part 2

(function, date, array, condition, loop)

JavaScript Function

- 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).
- 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: **{ }**

JavaScript Function - Syntax

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

- Function **parameters** are the **names** listed in the function definition.
- Function **arguments** are the real **values** received by the function when it is invoked.
- Inside the function, the arguments (the parameters) behave as local variables.

JavaScript Function Return

- When JavaScript reaches a **return statement**, the function will stop executing.
- If the function was invoked from a statement, JavaScript will "return" to execute the code after the invoking statement.
- Functions often compute a **return value**. The return value is "returned" back to the "caller":

JavaScript Function Return

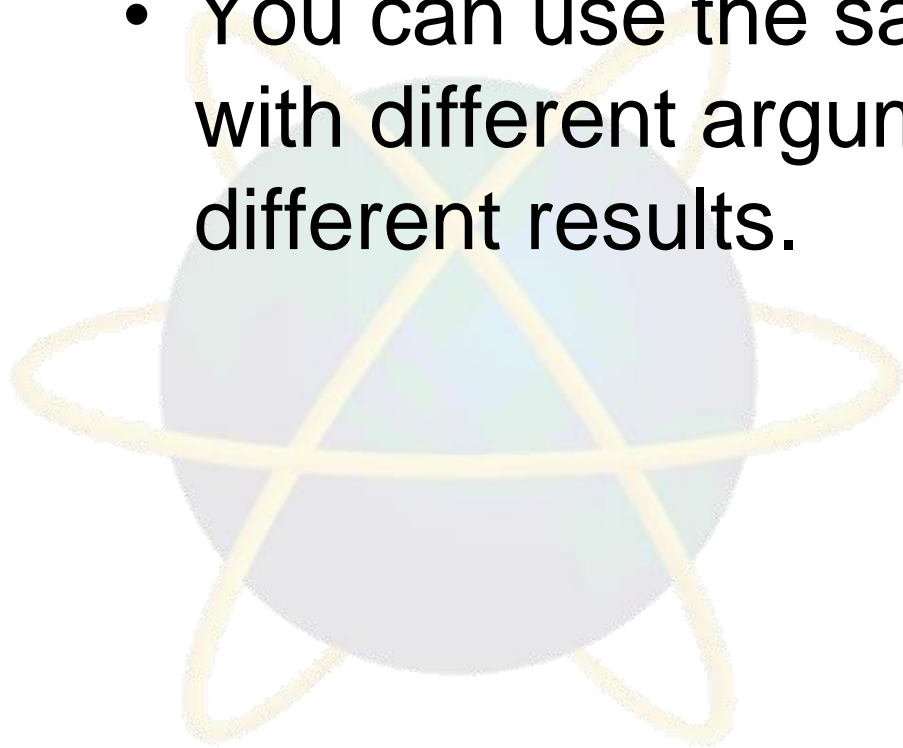
This example calls a function which performs a calculation:

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

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

Why use JavaScript Functions?

- You can reuse code: Define the code once, and use it many times.
- You can use the same code many times with different arguments, to produce different results.



Different ways of using function

- The () Operator Invokes the Function

```
function toCelsius(fahrenheit) {  
    return (5/9) * (fahrenheit-32);  
}  
document.getElementById("demo").innerHTML = toCelsius;
```

- Functions Used as Variable Values

```
var x = toCelsius(77);  
var text = "The temperature is " + x + " Celsius";  
  
var text = "The temperature is " + toCelsius(77) + " Celsius";
```

JavaScript Date()

- The Date object lets you work with dates (years, months, days, hours, minutes, seconds, and milliseconds).
- Date objects are created with the **new Date()** constructor.
- There are **4 ways** of initiating a date:
 - new Date()
 - new Date(milliseconds)
 - new Date(dateString)
 - new Date(year, month, day, hours, minutes, seconds, milliseconds)

JavaScript Date() – Time Zones

- When setting a date, without specifying the time zone, JavaScript will use the browser's time zone.
- When getting a date, without specifying the time zone, the result is converted to the browser's time zone.
- In other words: If a date/time is created in GMT (Greenwich Mean Time), the date/time will be converted to CDT (Central US Daylight Time) if a user browses from central US.

JavaScript Date Formats

- There are generally 4 types of JavaScript date input formats:

Type	Example
ISO Date	"2015-03-25" (The International Standard)
Short Date	"03/25/2015"
Long Date	"Mar 25 2015" or "25 Mar 2015"
Full Date	"Wednesday March 25 2015"

JavaScript Date Methods

- Get methods are used for getting a part of a date. Here are the most common Date Methods:

Method	Description
getDate()	Get the day as a number (1-31)
getDay()	Get the weekday as a number (0-6)
getFullYear()	Get the four digit year (yyyy)
getHours()	Get the hour (0-23)
getMilliseconds()	Get the milliseconds (0-999)
getMinutes()	Get the minutes (0-59)
getMonth()	Get the month (0-11)
getSeconds()	Get the seconds (0-59)
getTime()	Get the time (milliseconds since January 1, 1970)

JavaScript Arrays

- What is an Array?
- An array is a special variable, which can hold more than one value at a time.

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

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

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

JavaScript Conditions

- In JavaScript we have the following conditional statements:
- Use **if** to specify a block of code to be executed, if a specified condition is true.
- Use **else** to specify a block of code to be executed, if the same condition is false.
- Use **else if** to specify a new condition to test, if the first condition is false.
- Use **switch** to specify many alternative blocks of code to be executed.

JavaScript Conditions – if statement

- Use the **if** statement to specify a block of JavaScript code to be executed if a condition is true.



```
if (condition) {  
    block of code to be executed if the condition is true  
}
```

JavaScript Conditions – else Statement

- Use the **else** statement to specify a block of code to be executed if the condition is false.

```
if (condition) {  
    block of code to be executed if the condition is true  
} else {  
    block of code to be executed if the condition is false  
}
```

JavaScript Conditions – else if Statement

- Use the **else if** statement to specify a new condition if the first condition is false.

```
if (condition1) {  
    block of code to be executed if condition1 is true  
} else if (condition2) {  
    block of code to be executed if the condition1 is false and  
condition2 is true  
} else {  
    block of code to be executed if the condition1 is false and  
condition2 is false  
}
```


JavaScript Conditions – switch Statement

- Use the switch statement to select one of many blocks of code to be executed.

```
switch(expression) {  
  case n:  
    code block;  
    break;  
  case n:  
    code block;  
    break;  
  default:  
    default code block;  
}
```

- The switch expression is evaluated once.
- The value of the expression is compared with the values of each case.
- If there is a match, the associated block of code is executed.

JavaScript Loops

- Loops can execute a block of code a number of times.
- JavaScript supports different kinds of loops:
 - **for** - loops through a block of code a number of times
 - **for/in** - loops through the properties of an object
 - **while** - loops through a block of code while a specified condition is true
 - **do/while** - also loops through a block of code while a specified condition is true

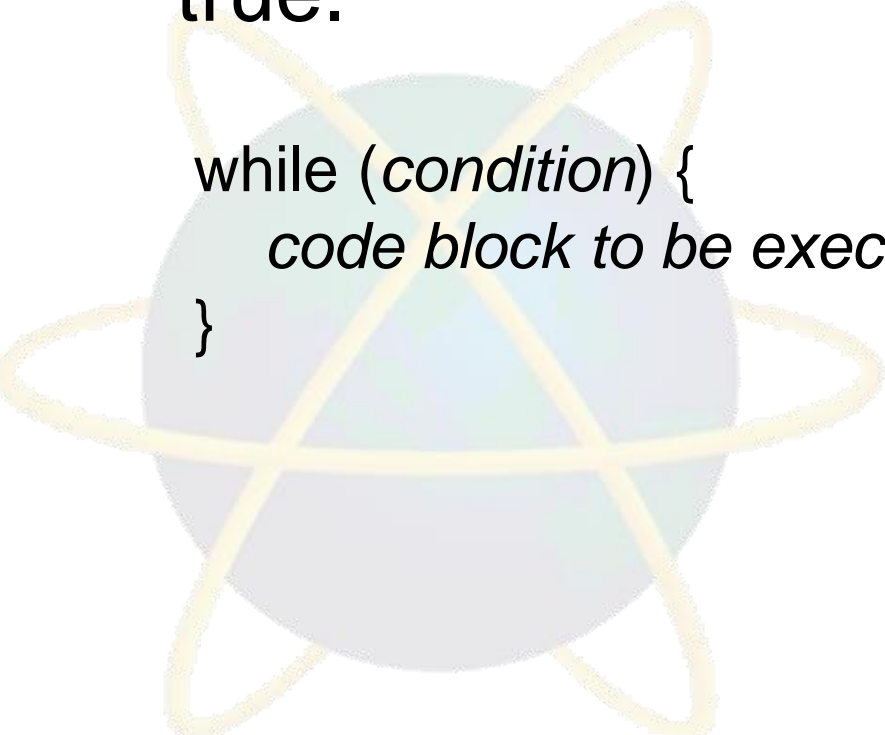
JavaScript Loops – for loop

- The for loop is often the tool you will use when you want to create a loop.
- The for loop has the following syntax:

```
for (statement 1; statement 2; statement 3) {  
    code block to be executed  
}
```

JavaScript Loops – while loop

- The while loop loops through a block of code as long as a specified condition is true.

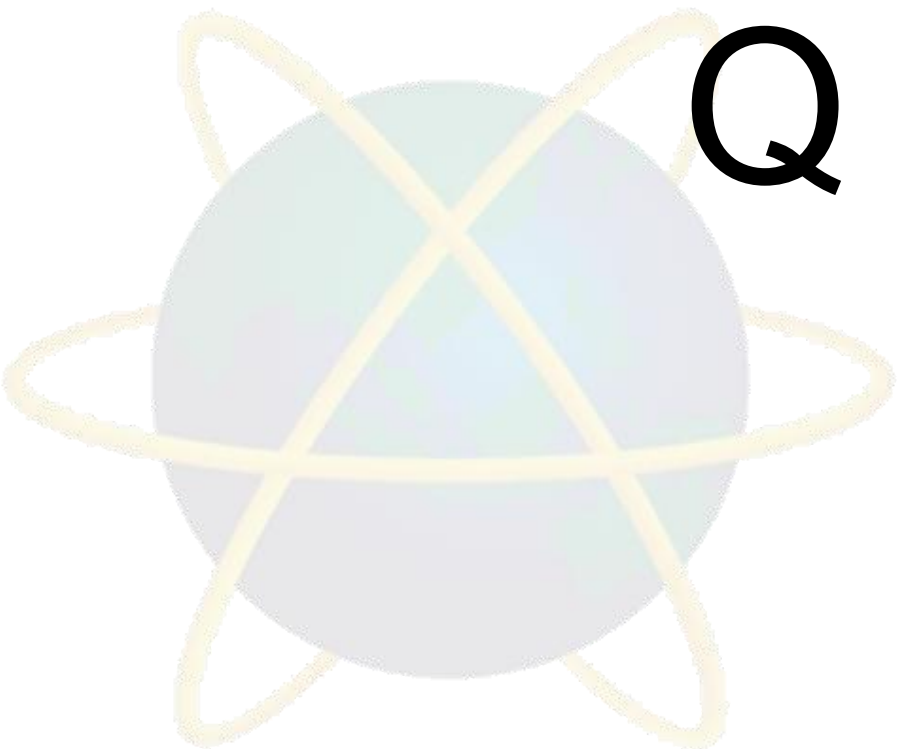


```
while (condition) {  
    code block to be executed  
}
```

JavaScript Loops – do/ while loop

- The do/while loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

```
do {  
    code block to be executed  
}  
while (condition);
```



Q & A