JavaScript

Document Object Model

Call Back Function

Functions and callbacks

- **Two ways to declare a function**

1. **Standard function declaration**

- We've already seen that functions can be declared using this syntax:

```
function functionName(parameters)
                { // code to be executed }
```

- Function Call:

```
functionName(parameters);
```

- Notice that we do not add a semicolon at the end of a function declaration.
  - Semicolons are used to separate executable JavaScript statements, and a function declaration is not an executable statement.

Example 1

```
<script>
function sum(a, b) {
    // this function returns a result
    return (a + b);   }

    function displayInPage(message, value) {
    // this function does not return anything
    document.body.innerHTML += message + value + "<br>“;   }

    var result = sum(3, 4);
    displayInPage("Result: ", result);

    // we could have written this
    displayInPage("Result: ", sum(10, 15));
</script>
```

Functions and callbacks

- 2-**Use a function expression**
- A JavaScript function can also be defined using an expression that can be stored in a variable.
- Then, the variable can be used as a function:

Example 1

- **var sum = function(a, b) {**
  return (a + b);
  };

  var displayInPage = function(message, value) {
  // this function does not return anything
  document.body.innerHTML += message + value + "
";
  };

  var result = sum(3, 4);
  displayInPage("Result: ", result);

  // we could have written this
  displayInPage("Result: ", sum(10, 15));

Example 1

- In previous example, the sum and displayInPage functions have been declared.
- We used a variable to store the function expression, then we can call the functions using the variable name.
- And we added a semicolon at the end, since we executed a JavaScript instruction, giving a value to a variable.
- The "function expression" is an "anonymous function", a function without a name, that represents a value that can be assigned to a variable. Then, the variable can be used to execute the function.

Functions and callbacks

- We say that functions are "first class objects" which can be manipulated like any other object/value in JavaScript.

- This means that functions can also be used as parameters to other functions. In this case they are called "callbacks".

Functions and callbacks

- **Callbacks**
- Indeed, as functions are first-class objects,
  - we can pass a function as an argument, as a parameter to another function and later execute that passed-in function or even return it to be executed later.
  - When we do this, we talk about *callback functions* in JavaScript: a function passed to another function, and executed inside the function we called.

Click Here!

Click Here!

Button Clicked Version 1
Button Clicked Version 2

Example 2 -HTML

- <!doctype html>
  <html>
  <head lang="en">
  <title>Function part 1</title>
  <meta charset="utf-8">
  </head>
  <body>
  ** Click in the page! **
  </body>
  </html>

Example 2 - JavaScript

- // Add a click **event listener on the whole document**
  // the processClick function is a callback:
  // a function called by the browser when a click event is detected

- **window.addEventListener('click', processClick);**

  function processClick(event) {
  document.body.innerHTML += "Button clicked
";
  }

  // We could have written this, with the body of the callback as an argument of the addEventListener function

- **window.addEventListener('click', function(evt) {**
  document.body.innerHTML += "Button clicked version 2
";
  });

Example 2

- The addEventListener function is one possible syntax for registering a function to be called when a given type of event occurs.

- The event object is the only parameter passed to event listeners

Output

Click Here!

Button Clicked Version 1
Button Clicked Version 2

Adding an event listener to specific HTML elements

- Instead of listening to event on the whole document (using addEventListener is the same as using window.addEventListener), we can listen to specific DOM elements.

Example 3

- For example, here is how we can listen to clicks on a specific button (whereas clicks on the rest of the document will be ignored).

Example – 3

- <!DOCTYPE html>
  <html lang="en">
  <body>
  <button id="myButton">Click me!</button>

  <script>
  var b = document.querySelector("#myButton");
  b.addEventListener('click', function(evt) {
  alert("Button clicked");
  });
  </script>
  </body>
  </html>

Dealing with key events

- When you listen to keyboard related events (keydown, keyup or keypressed), the event parameter passed to the listener function will contain the code of the key that fired the event.
- Then it is possible to test which key has been pressed or released,

```
window.addEventListener('keydown', function(event) {
  if (event.keyCode === 37) {
    //left arrow was pressed
  }
});
```

Key	Code
backspace	8
tab	9
enter	13
shift	16
ctrl	17
alt	18
pause/break	19
caps lock	20
escape	27
(space)	32
page up	33
page down	34
end	35
home	36
left arrow	37
up arrow	38
right arrow	39
down arrow	40
insert	45
delete	46
0	48
1	49
2	50
3	51
4	52
5	53
6	54
7	55
8	56
9	57
a	65
b	66
c	67
d	68

Key	Code
e	69
f	70
g	71
h	72
i	73
j	74
k	75
l	76
m	77
n	78
o	79
p	80
q	81
r	82
s	83
t	84
u	85
v	86
w	87
x	88
y	89
z	90
left window key	91
right window key	92
select key	93
numpad 0	96
numpad 1	97
numpad 2	98
numpad 3	99
numpad 4	100
numpad 5	101
numpad 6	102
numpad 7	103

Key	Code
numpad 8	104
numpad 9	105
multiply	106
add	107
subtract	109
decimal point	110
divide	111
f1	112
f2	113
f3	114
f4	115
f5	116
f6	117
f7	118
f8	119
f9	120
f10	121
f11	122
f12	123
num lock	144
scroll lock	145
semi-colon	186
equal sign	187
comma	188
dash	189
period	190
forward slash	191
grave accent	192
open bracket	219
back slash	220
close braket	221
single quote	222

Example 4

- Print on the fly as the user types in a text input field