Event-driven programming & Callback Function

Event driven programming

Event-Driven Programming makes use of the following concepts:

- an event loop (event listener) - it keeps listening for event triggers and when an event occurs, the event loop determines the event callback (event handler) and invokes it
- an event handler that gets called when an event is triggered, and performs the said action

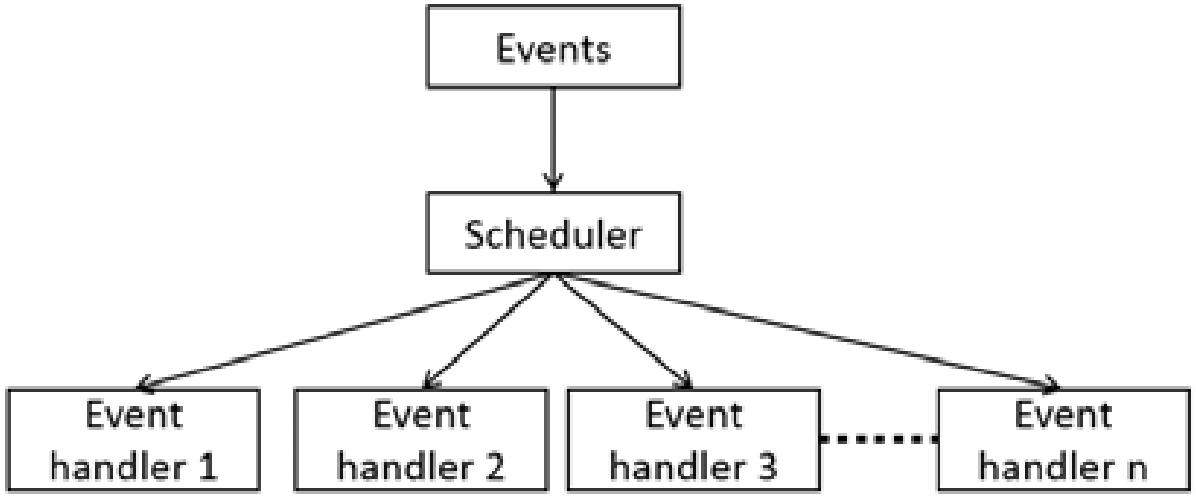


Image Courtesy: https://www.researchgate.net/figure/Architecture-of-Event-driven-Programming-Model_fig1_323279886

Event driven programming

- In event-driven paradigm there are two actors –

  the subject (event emitter) and the observer (event listener)

- On detecting an event, the event listener triggers a callback function (event handler), which performs the corresponding action

  Eg. - Clicking (*event*)

  a “print” button (*event listener*)

  activates the actual print process (*event handler*)

- In this paradigm, flow of execution is dictated by events

  The program loads and then waits for user input events

  As each event occurs, the program runs a particular code in response

  The overall flow of what code is executed is determined by the series of events that occur

- Most useful for asynchronous execution

How event-driven approach applies to JS in the browser?

- JS engine running in the browser provides an event-driven platform for JS
- JS in the browser interacts with event emitters (HTML elements capable of emitting events)
- JS functions act as event listeners and handle/ react to events emitted by the elements

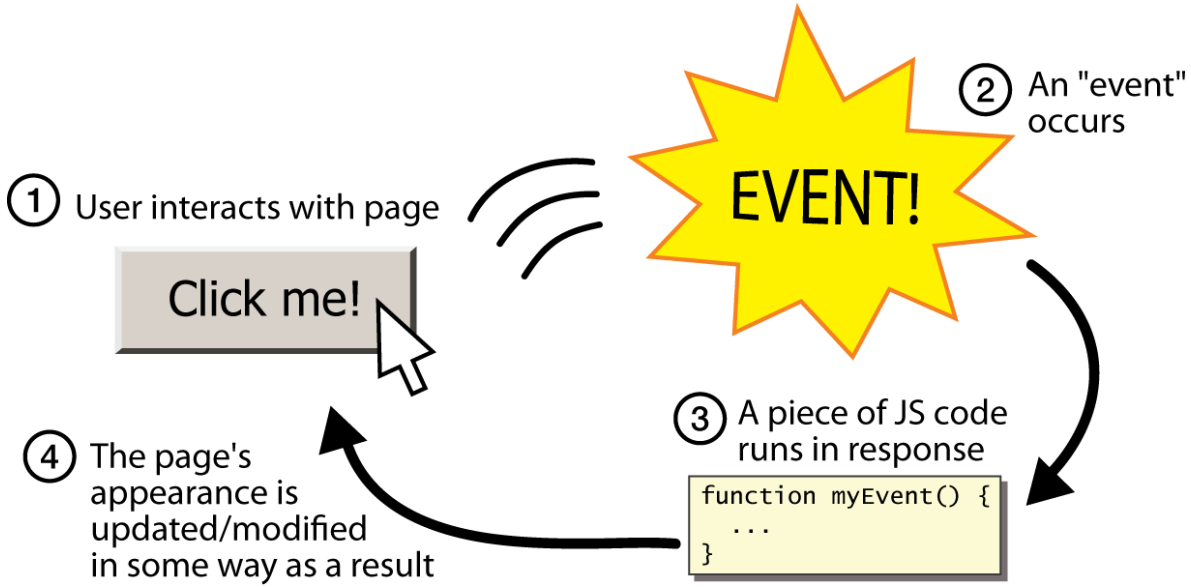


Image Courtesy: https://courses.cs.washington.edu/courses/cse331/13sp/lectures/lect24-events.pdf

JavaScript Events

What is an *Event* ?

something the browser or user does, like clicking a button, loading of page, change in input field value, occurrence of an error

Types of events

i. User Interface events (load, unload, error, resize, scroll)
i. Focus & Blur events (focus, blur, focusin, focusout)
ii. Mouse events (click, mouse-down, mouseup, mouseout, mouseover)
iii. Keyboard events (input, keydown, keypress, keyup)
iv. Form events (submit, input, change)
v. Mutation events
vi. HTML5 events
vii. CSS events

Callback Function

In technical parlance

In JavaScript, functions are objects. Because of this, functions can take functions as arguments, and can be returned by other functions. Functions that do this are called higher-order functions. Any function that is passed as an argument and subsequently called by the function that receives it, is called a callback function.

In simple words

A callback is a function that is to be executed after another function has finished executing - hence the name ‘call back’.

Callback functions allow the scheduling of asynchronous actions, i.e., actions that we initiate now, but they finish later

Text Courtesy: https://www.sitepoint.com/callbacks-javascript/

Callback Function

```
// sequential processing without callback
function add(n1, n2)
{
    let y = n1 + n2;
    console.log("Sum = " + y);
}

function multiply(n1, n2)
{
    let y = n1 * n2;
    console.log("Product = " + y);
}

function lastFunc()
{
    console.log("Calculation Done");
}

add(2, 3);
multiply(2,3);
lastFunc();
```

Sum = 5
Product = 6
Calculation Done

With call back

```
function add(n1, n2)
{
    let y = n1 + n2;
    console.log("Sum = " + y);
}

function multiply(n1, n2)
{
    let y = n1 * n2;
    console.log("Product = " + y);
}

function lastFunc()
{
    console.log("Calculation Done");
}

setTimeout(add, 2000, 2, 3);
multiply(2,3);
lastFunc();
```

Product = 6
Calculation Done
Sum = 5

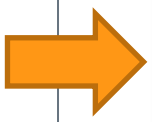
Callback Function

```
// Using Callback
function add(n1, n2)
{
    let y = n1 + n2;
    console.log("Sum = " + y);
    lastFunc();
}

function multiply(n1, n2)
{
    let y = n1 * n2;
    console.log("Product = " + y);
}

function lastFunc()
{
    console.log("Calculation Done");
}

setTimeout(add, 2000, 2, 3);
multiply(2,3);
```



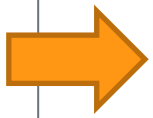
```
Product = 6
Sum = 5
Calculation Done
```

Callback as an Anonymous Function

```
// Anonymous Function
function multiply(n1, n2)
{
    let y = n1 * n2;
    console.log("Product = " + y);
}

function lastFunc()
{
    console.log("Calculation Done");
}

setTimeout(function add(n1, n2) {
    let y = n1 + n2;
    console.log("Sum = " + y);
    lastFunc();
}, 2000, 2, 3);
multiply(2,3);
```



```
Product = 6
Sum = 5
Calculation Done
```