# Events

## What are Events

A sidemenu sliding out upon clicking.

A form-field turning red when the user types a wrong format.

**Listening** – waiting for the user to perform a particular action before responding accordingly.

## Event Driven Programming

* JS isn’t executed line by line then forgotten.
* After a browser loads HTML and CSS, an *intepreter* is used to run Javascript.
* Though the code generally run from top to bottom, as developers we have no idea which code will actually be executed.
* We need to write code that will execute **asynchronously** (outside of the typical top-to-bottom flow.
* JS loads an lives in the background, **listening** for event triggers programmed.

## Asynchronicity

* Event-Driven Programming, where the flow of a program is driven by events.
* Each event triggers a function

## Setting up Event Handler

Here is the syntax:

element.addEventListener('nameOfEvent', functionToRun);

* element is the DOM node we want to tie the event to.
* addEventListener is the *method*.
* nameOfEvent is the event we want to listen for.
* functionToRun is the name of the function we want to run when the event occurs.
  + functionToRun has **no parentheses**!!! functionToRun NOT functionToRun()
  + Specifying a function as an argument to another function, like we have done in this example, is called a callback function.

Setting up:

1. Set up the function
2. Set up the DOM Node
3. Add Event Listener

var alertUser = function() {

alert(‘Button has been clicked.’);

}

var button = document.querySelector(‘button’);

button.addEventListener(‘click’,alertUser);

## Types of Events

* click
* keydown – when user first press a key
* keyup – when user releases a key
* focus – selects on a field
* blur – moves away from a field
* submit – submit a form
* load – when page has finished loading
* resize – when browser has been resized
* scroll – when user scrolls up and down
* mouseenter – when mouse enters an element
* mouseleave – when mouse leaves an element

## Event Listeners

document.querySelector(‘.circle’).addEventListener(‘click’,turnRed)

var turnRed = function() {

**this**.style.backgroundColor = “red”;

}

We is ‘**this**’ so that the function acts on the DOM node.

Typically, this refers to the object that “owns” the function.

However, when the callback function is executed within the context of an event handler, ‘**this**’ refers to the DOM node which owns the context.

This allows us to execute function on a particular element.

## Event Listeners: case study

Can be used on an input search dropdown.

So the event listener is the input – after the user inputs the function is activated.

When the user has typed three letters, it will start search through the database for possible matches, and then display a dropdown for all those search matches.

## Case Study Exercise Solution

function testJS() {

console.log('working fine');

}

function addItem() {

var newItem = document.createElement('li'); **//li is the tag name//**

newItem.textContent**(or innerHTML)** = document.querySelector('input').value;

document.getElementById('todo-list').appendChild(newItem);

}

var button = document.querySelector('button');

button.addEventListener('click',addItem);

function printList() {

for (let i =0; i < document.getElementsByTagName('li').length; i++) {

console.log(document.getElementsByTagName('li')[i])

}

}