All about Javascript

Kiran Pachhai

Functions

- Blocks of code that can be executed whenever they are called.
- Used to define reusable code that can be called multiple times.
- Can take multiple arguments and return multiple values.

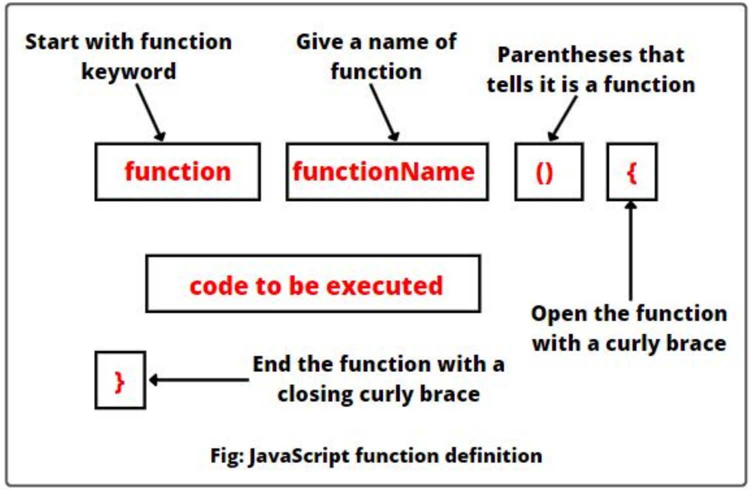
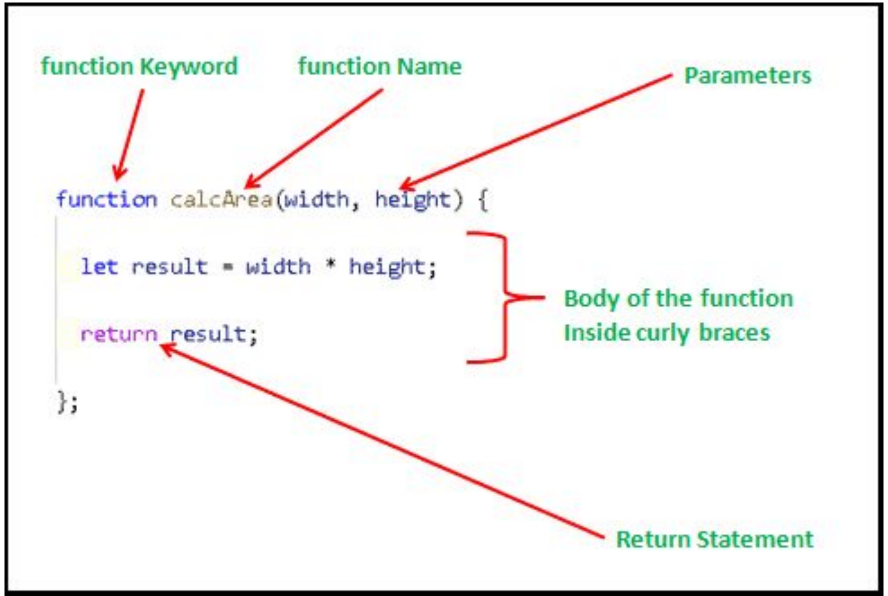

Start with function keyword
Give a name of function
Parentheses that tells it is a function
function
functionName
()
{
code to be executed
Open the function with a curly brace
}
End the function with a closing curly brace

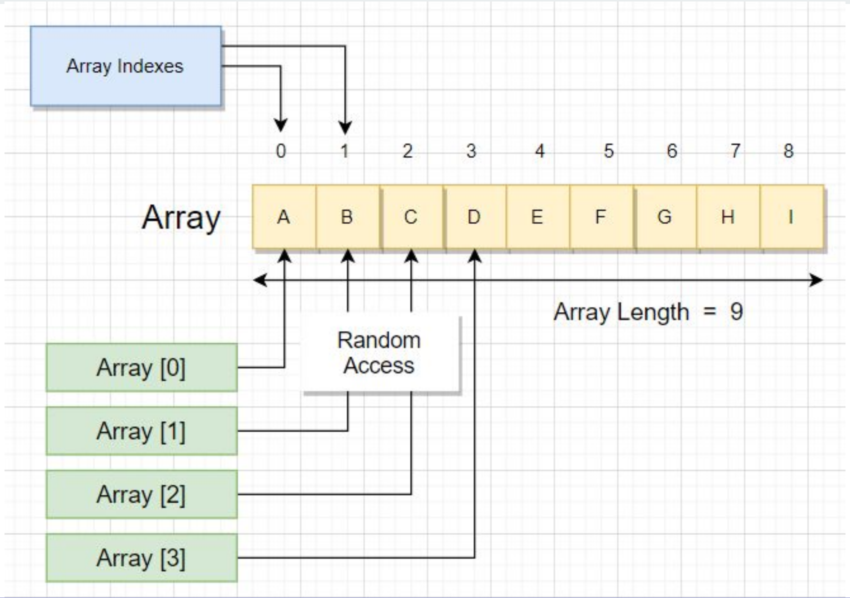
Fig: JavaScript function definition

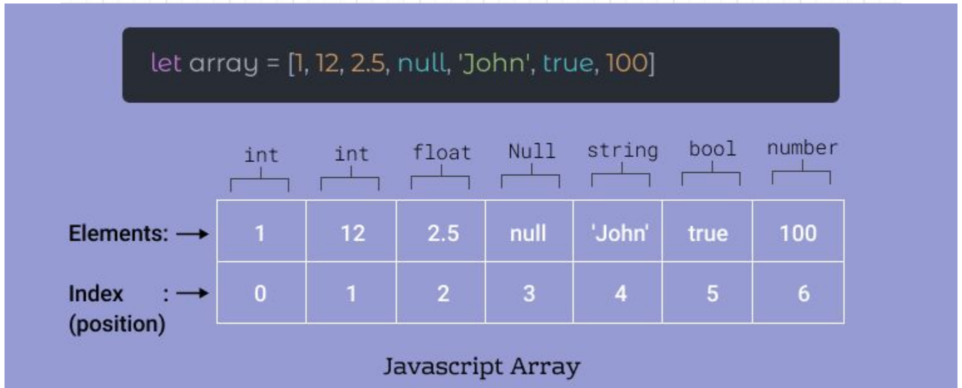


```
function calcArea(width, height) {

  let result = width * height;

  return result;

};
```

Arrays

- An array is a special type of object used to store a collection of values.
- Can access individual elements of an array using their index number, which is the position of the element in the array.
- Indices are zero-based, meaning the first element is at index 0, the second is at index 1, etc.
- Can add new elements to an array using the push() method, and remove elements from an array using the pop() method.

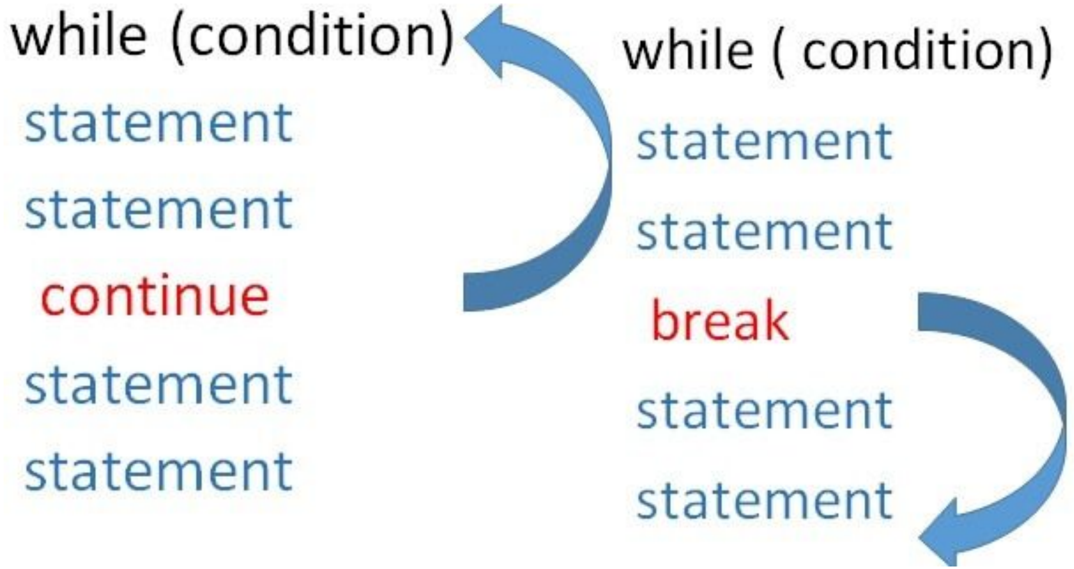


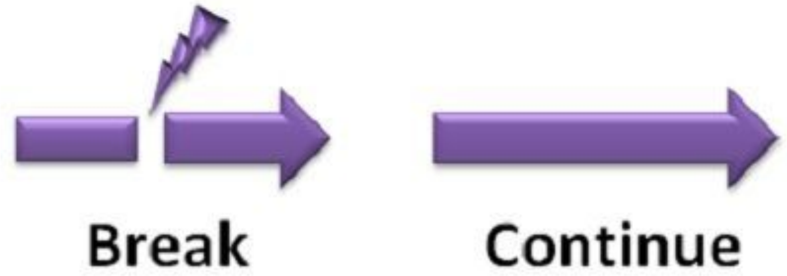


```
let array = [1, 12, 2.5, null, 'John', true, 100]
```

int int float Null string bool number
Elements: → 1 12 2.5 null 'John' true 100
Index (position) : → 0 1 2 3 4 5 6
Javascript Array

Break v continue v return

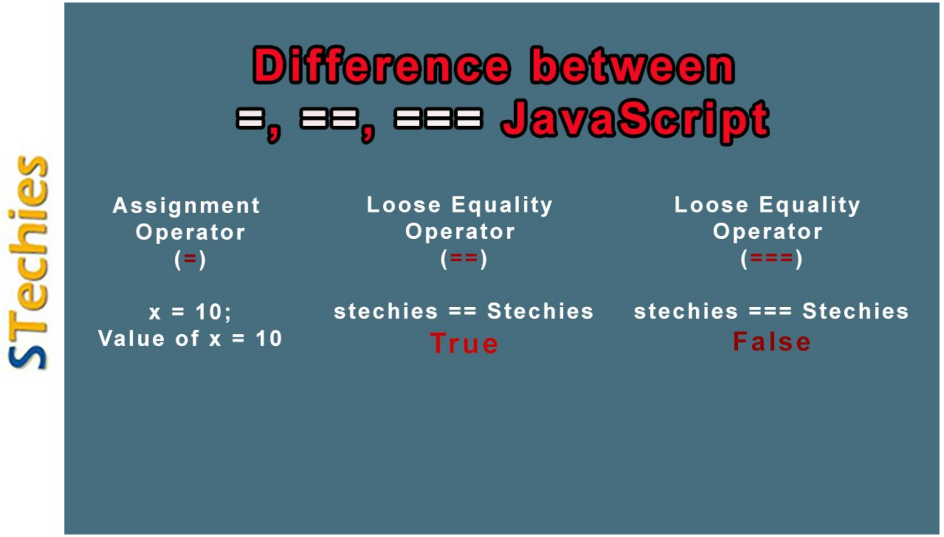




- The break and continue statements are used to control the flow of a loop.
- The break statement is used to immediately exit the loop, while the continue statement is used to skip the current iteration of the loop and continue with the next iteration.
- The return statement is used to return a value from a function. When a return statement is executed, the function stops executing and returns the specified value.

Equality check

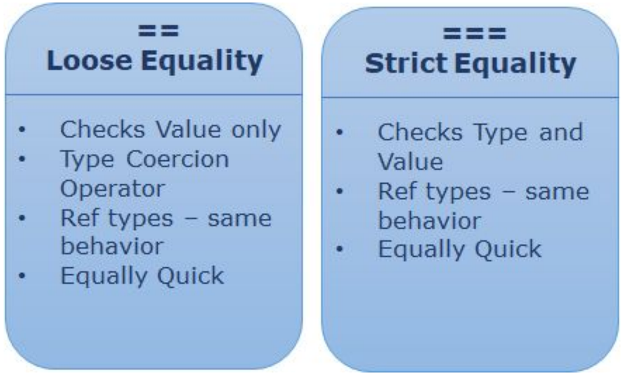
- There are two types of equality checks: strict equality (===) and loose equality (==).
- Strict equality checks whether the two operands have the same type and value.
- Loose equality checks whether the two operands have the same value after they are converted to the same type.
- It is generally recommended to use strict equality when comparing values in JavaScript, and to use loose equality sparingly, if at all.



STechies

Difference between =, ==, === JavaScript

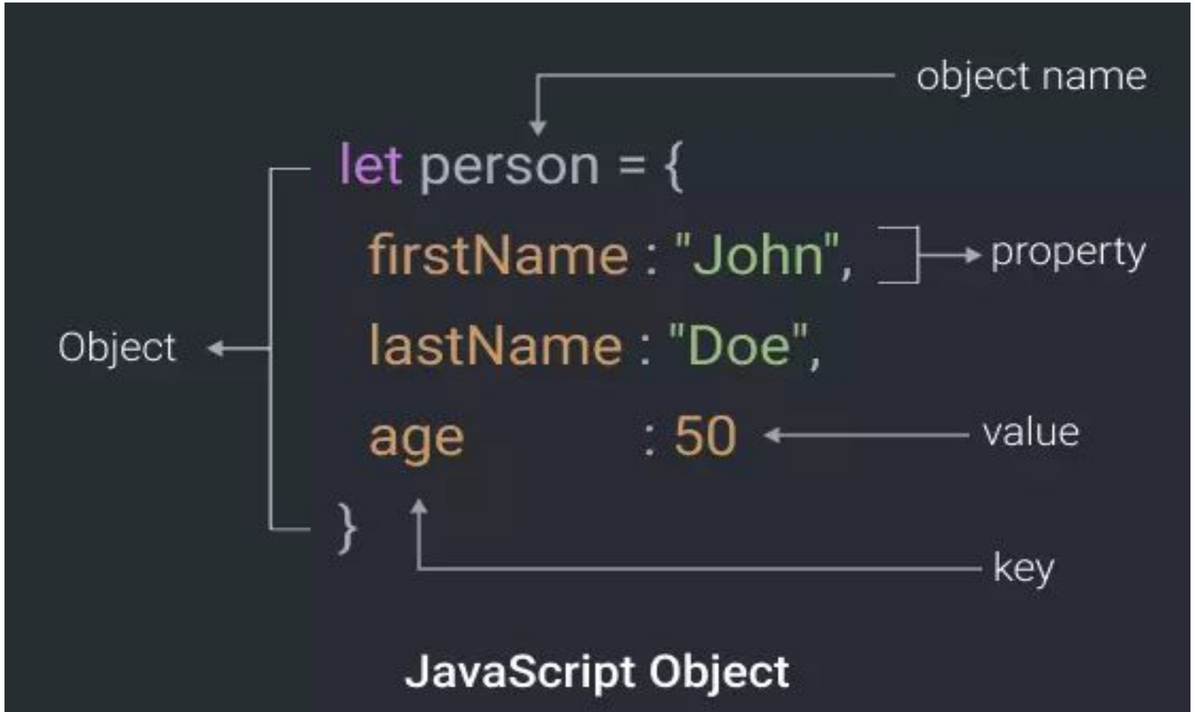
Assignment Operator (=)	Loose Equality Operator (==)	Loose Equality Operator (===)
x = 10; Value of x = 10	stechies == Stechies True	stechies === Stechies False



== Loose Equality	=== Strict Equality
• Checks Value only • Type Coercion Operator • Ref types – same behavior • Equally Quick	• Checks Type and Value • Ref types – same behavior • Equally Quick

Objects

- An object in JavaScript is a collection of key-value pairs
- Similar to a dictionary in Python or a Map in Java.
- Each property of an object has a name and a value
- Properties can be added or removed from an object using dot notation (object.property) or square bracket notation (object["property"])
- Objects can also have methods
- JavaScript objects can be thought of as a special type of associative array, where the keys can be any string or symbol (not just numbers, as in regular arrays).
- The Object.keys() and Object.values() methods can be used to get the keys and values of an object



```
let person = {
  firstName : "John",
  lastName : "Doe",
  age       : 50
}
```

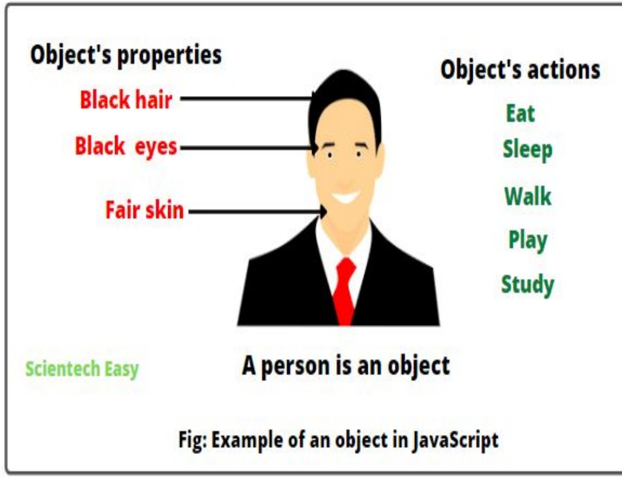
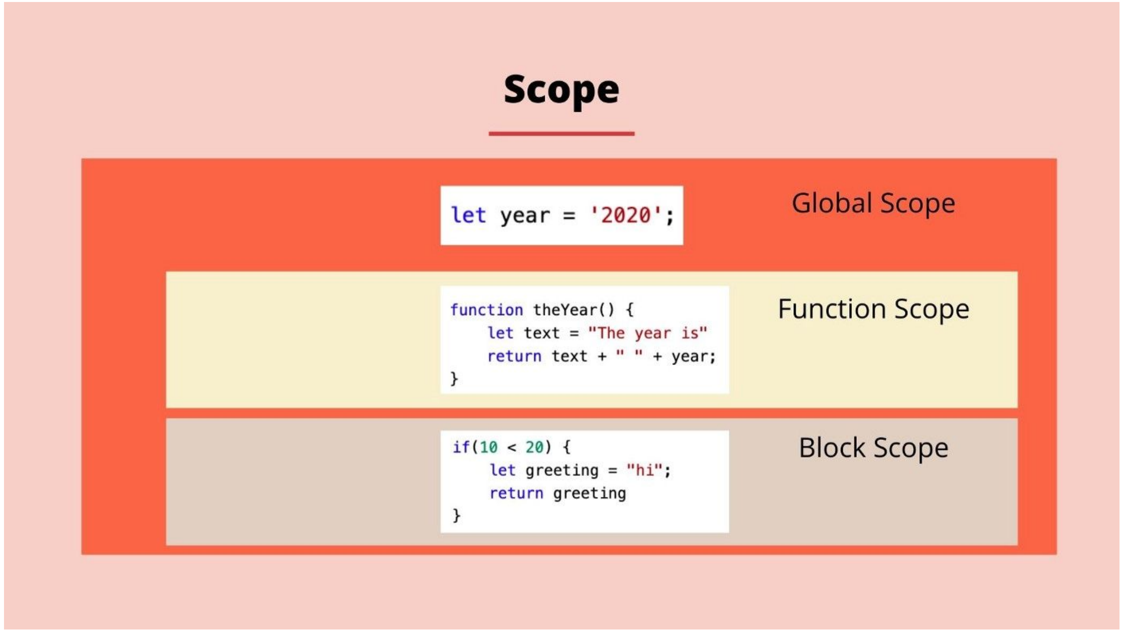


Fig: Example of an object in JavaScript

Scoping

- Scope refers to the visibility and accessibility of variables and functions in different parts of your code.
- Two types of scopes: global scope and local scope.
- Global scope: can be accessed from anywhere in your code.
- Local scope: can only be accessed within the context(such as a function)



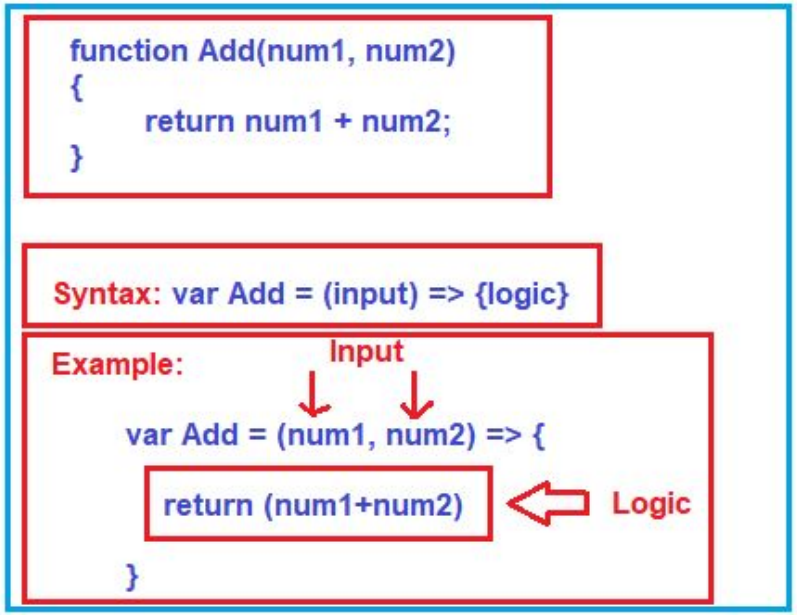
```
let year = '2020';
```

```
function theYear() {
    let text = "The year is"
    return text + " " + year;
}
```

```
if(10 < 20) {
    let greeting = "hi";
    return greeting
}
```

Arrow functions

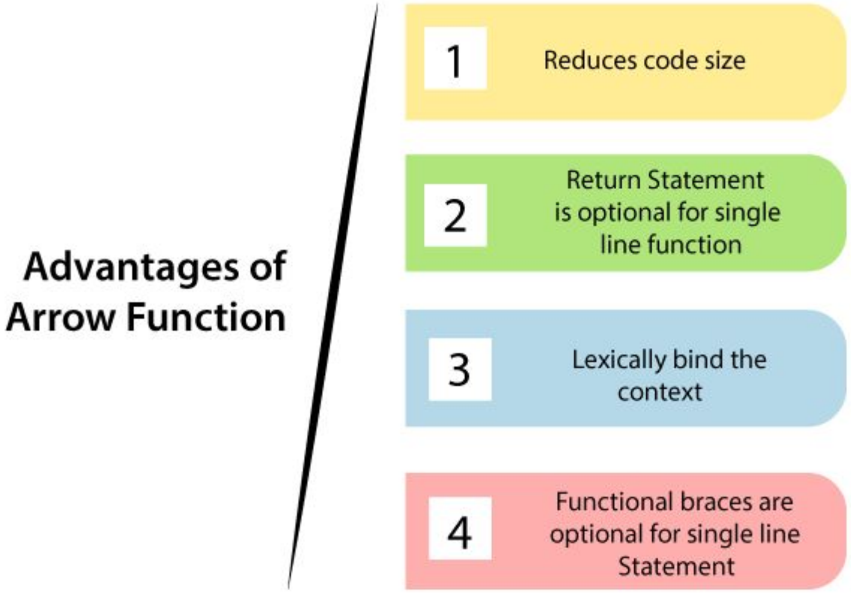
- JavaScript has arrow functions, which are defined using the => syntax.
- Arrow functions are shorter and more concise than regular functions.
- Arrow functions don't have their own this value.
- The this value in an arrow function is determined by the surrounding context in which the function is defined.
- This is different from regular functions, where the this value is determined by how the function is called.



```
function Add(num1, num2)
{
        return num1 + num2;
}
```

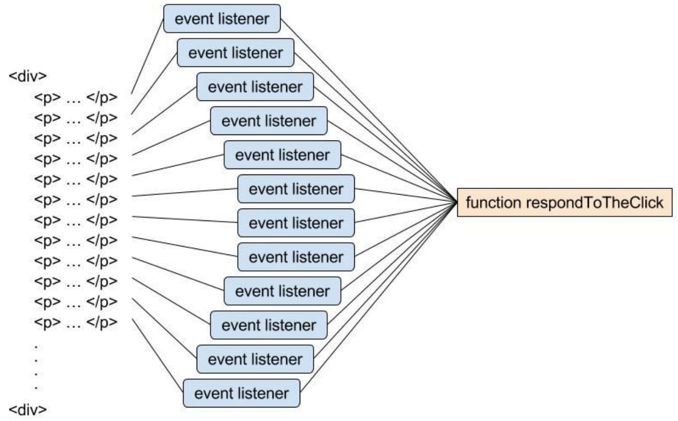
```
Syntax: var Add = (input) => {logic}
```

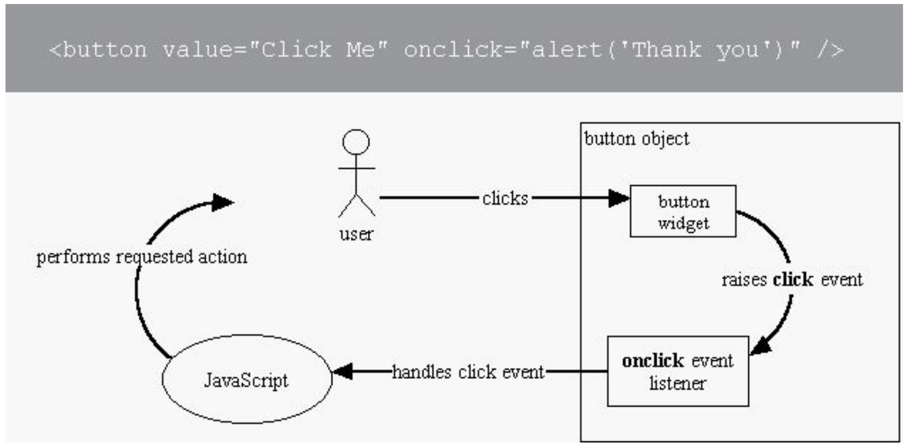
```
Example:          Input

     var Add = (num1, num2) => {

        return (num1+num2)      Logic

     }
```


Advantages of Arrow Function
1 Reduces code size
2 Return Statement is optional for single line function
3 Lexically bind the context
4 Functional braces are optional for single line Statement

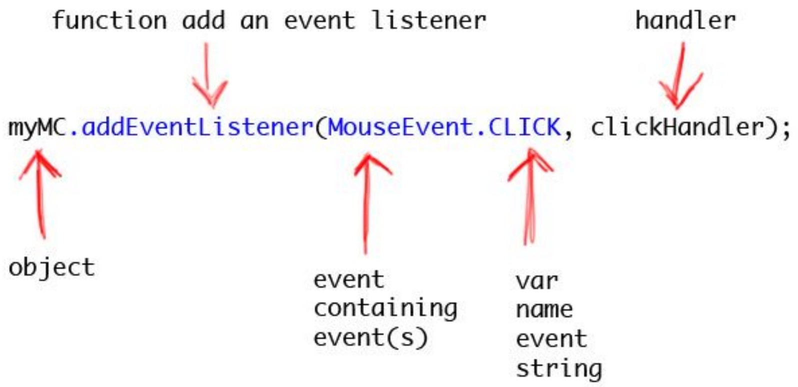
Event Listeners

- A function that is called when a specific event occurs on a webpage
- Examples of events include clicking a button or hovering over an element
- There are many different types of events that you can listen for in JavaScript



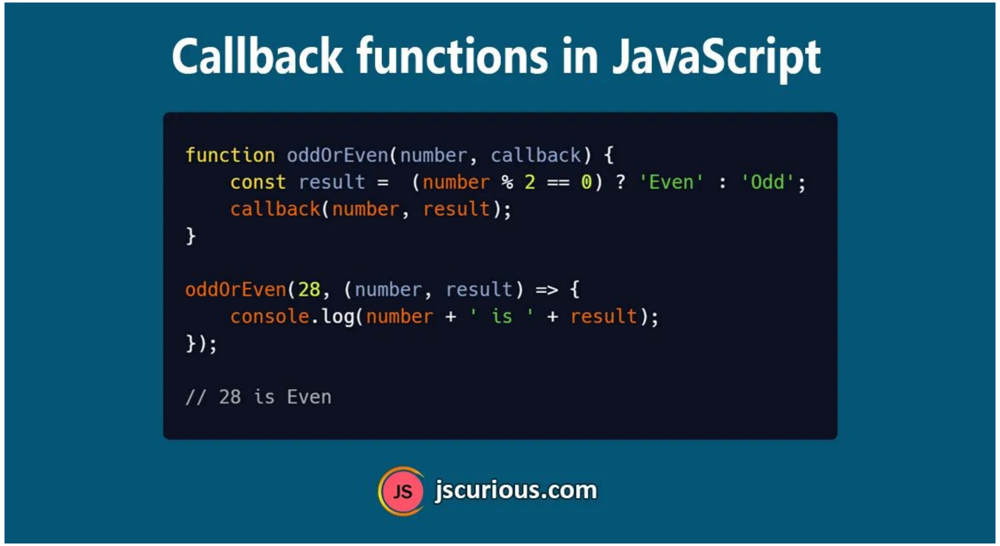


```
<button value="Click Me" onclick="alert('Thank you')" />
```



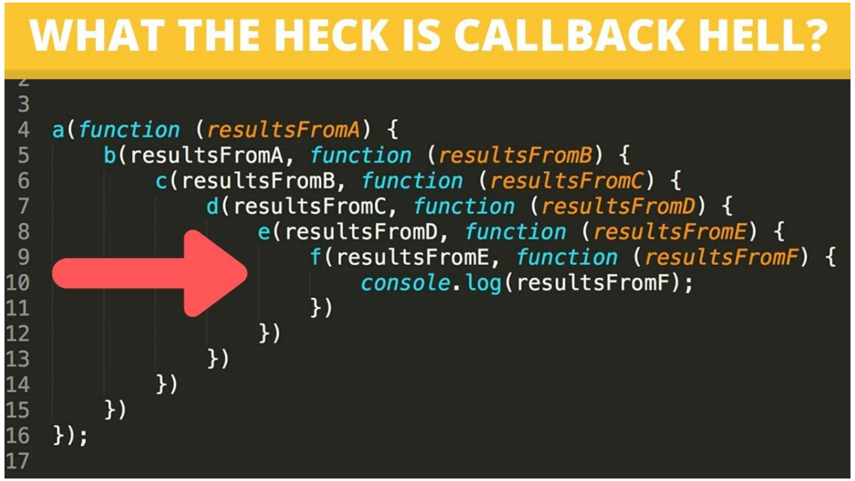
```
myMC.addEventListener(MouseEvent.CLICK, clickHandler);
```

Callbacks


Callback functions in JavaScript

```
function oddOrEven(number, callback) {
    const result =  (number % 2 == 0) ? 'Even' : 'Odd';
    callback(number, result);
}

oddOrEven(28, (number, result) => {
    console.log(number + ' is ' + result);
});

// 28 is Even
```

jscurious.com

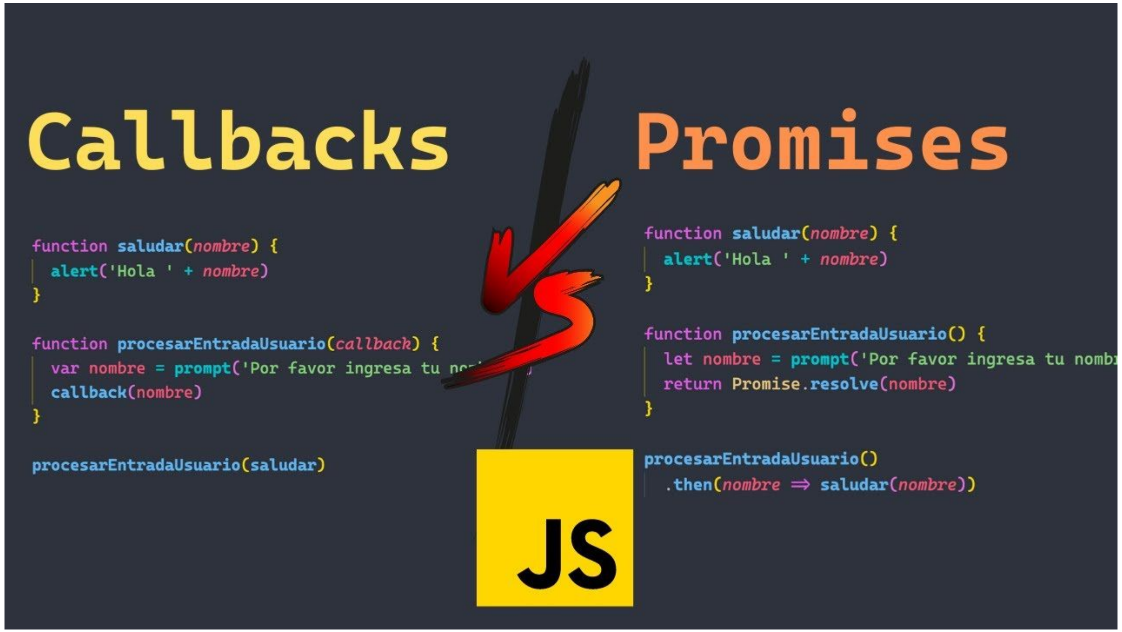

WHAT THE HECK IS CALLBACK HELL?

```
a(function (resultsFromA) {
    b(resultsFromA, function (resultsFromB) {
        c(resultsFromB, function (resultsFromC) {
            d(resultsFromC, function (resultsFromD) {
                e(resultsFromD, function (resultsFromE) {
                    f(resultsFromE, function (resultsFromF) {
                        console.log(resultsFromF);
                    })
                })
            })
        })
    })
});
```

- A callback is a function passed as an argument to another function
- Callbacks are executed after some kind of event occurs
- This allows for asynchronous behavior in code
- Callbacks are commonly used in JavaScript to handle asynchronous events, such as user clicks on a web page

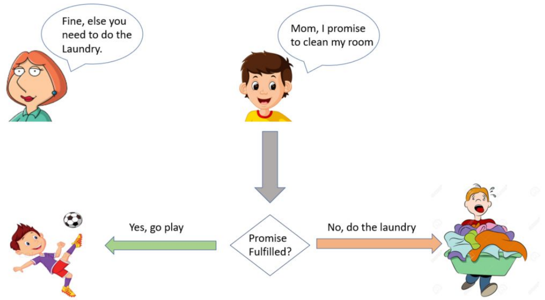
Promises and Fetch

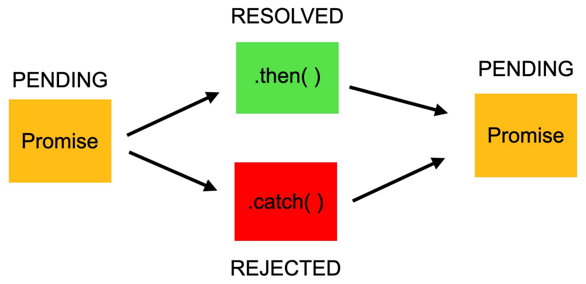
- Promises and fetch are two related JavaScript concepts
- A promise is an object that represents the result of an asynchronous operation
- Fetch is a JavaScript API for making network requests and retrieving data asynchronously
- The fetch function returns a promise, which can be used with the then method to parse and handle the server response



```
function saludar(nombre) {
  alert('Hola ' + nombre)
}

function procesarEntradaUsuario(callback) {
  var nombre = prompt('Por favor ingresa tu no
  callback(nombre)
}

procesarEntradaUsuario(saludar)
```

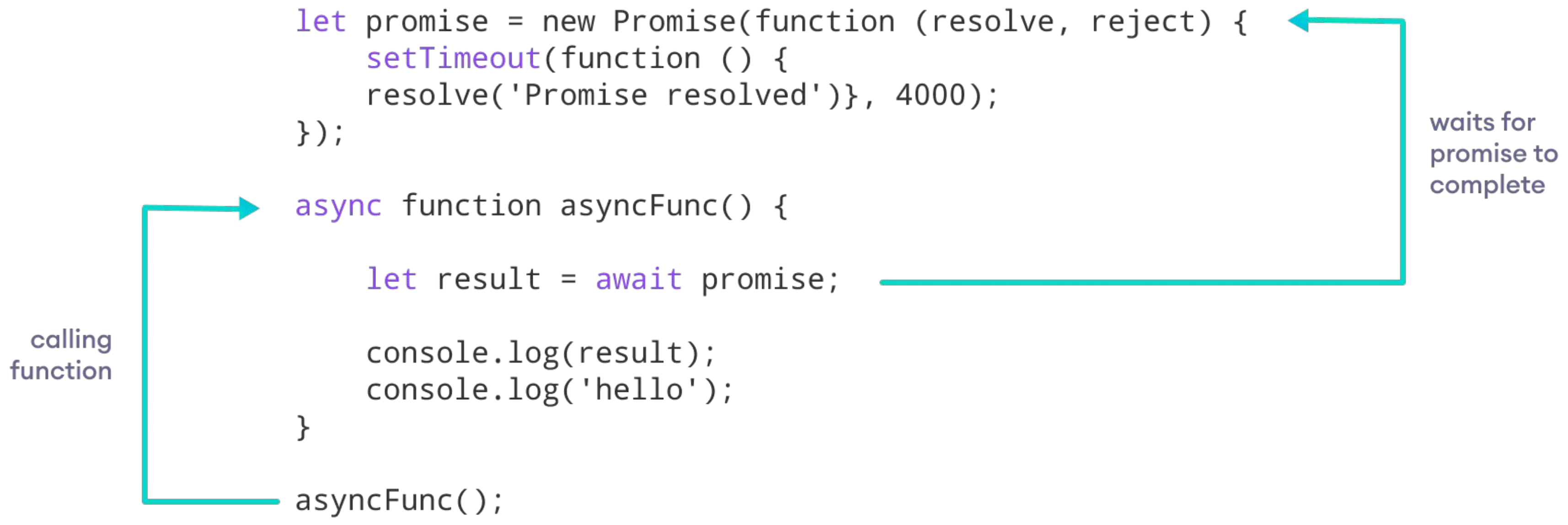
```
function saludar(nombre) {
  alert('Hola ' + nombre)
}

function procesarEntradaUsuario() {
  let nombre = prompt('Por favor ingresa tu nomb
  return Promise.resolve(nombre)
}

procesarEntradaUsuario()
  .then(nombre => saludar(nombre))
```


Fine, else you need to do the Laundry.
Mom, I promise to clean my room
Yes, go play
Promise Fulfilled?
No, do the laundry



Await

- Used in JavaScript to wait for a promise to be resolved or rejected
- await can only be used inside an async function
- await causes the function to pause execution until the promise is either resolved or rejected



```
let promise = new Promise(function (resolve, reject) {
    setTimeout(function () {
    resolve('Promise resolved')}, 4000);
});

async function asyncFunc() {

    let result = await promise;

    console.log(result);
    console.log('hello');
}

asyncFunc();
```