It means that you can treat functions same way as you would treat an object, a number or a string. You can assign a function as a value to a variable. Example:

**var** aNumber = 1;

**var** aString = 'Just a string';

**var** aNotSoUsefulFunction = **function** ()

{

console.log('Hello, World!');

}

You can assign a function as a value to a variable, just like you can assing a number or a string as a value to a variable.

what does it have to do with callbacks?

**functions are first class objects in Javascript.**

**we can pass them as arguments to another function.**Same as you can pass a number or a string as an argument to a function.

 Javascript is a dynamically typed language , don’t worry about string or int datatype parameters

**function** **printGreeting**(name) {

console.log('Hello', name);

}

printGreeting('Jerry');

printGreeting(123);

we pass another function as the name argument.

|  |  |  |
| --- | --- | --- |
| **function** **printGreeting**(  **function** () {  *// this function doesn't do anything*  }  )  {  console.log('Hello', name);  } | **Same code can be rewritten as:** Replace **name with function** **emptyFunction** () {*// this function doesn't do anything}*  **function** **printGreeting**(name) {  console.log('Hello', name);  }  printGreeting(**function** **emptyFunction**() {  *// this function doesn't do anything*  }); | In fact, we **do not even need to name** our emptyFunction here. So the code can be again rewritten as:  **function** **printGreeting**(name) {  console.log('Hello', name);  }  printGreeting(**function** () {  *// this function doesn't do anything*  });  Such functions are called **anonymous functions** in Javascript and you will see them **everywhere** in any Javascript code. |
| **function** **myMethodToPrint**(functionToExecute, argumentToPass) {  **for** (**let** i=1; i<=10; i++) {  console.log(i);  }  functionToExecute(argumentToPass);  }  **function** **printGreeting**(name) {  console.log('Hello', name);  }  myMethodToPrint(printGreeting, ‘Jerry');  **function** **printSquareOfNumber**(n) {  console.log(n\*n);  }  myMethodToPrint(printSquareOfNumber, 7); | | What we are doing here is that we are passing a function functionToExecute (example printGreeting) to another function myMethodToPrint and we are executing our functionToExecute inside the myMethodToPrint.  The **functionToExecute here represents a callback function**. So, our printGreeting and printSquareOfNumber are both callback functions. |
| Events  Callbacks are a must in an event listener.  We need to make use of the addEventListner method and append to the button DOM (HTML element). The addEventListener takes in two arguments. The first argument is a type of event we want to listen for and the second argument is a callback function to execute after the specified event is triggered.  **<button id="button">Who am I?</button>**  **const button = document.getElementById("button");**  **function introduce() {**  **console.log("I am a tomato!");**  **}**  **button.addEventListener("click", introduce); // I am a tomato!** | | Alternatively, you can directly insert an anonymous function as a callback instead.  const button = document.getElementById("button");  button.addEventListener("click", function() {  console.log("I am a tomato!");  }); // I am a tomato |