MODERN JavaScript

Karthik Raman

Variables Scope

Clayfin

- ❖ How to define variables in Javascript
  3 ways : var , let , const
- ❖ let and const in ES6 or ECMA2015

- ▪ ***var*** *keyword declares a variable which is scoped to its current execution context, optionally initializing to a value*
- ▪ ***let*** *keyword declares a block scoped variable, optionally initializing it to a value*
- ▪ ***const*** *keyword declares constants which are block scoped much like variables defined using* ***let*** *but the value of a constant cannot change. The* ***const*** *declaration creates a read-only reference to a value*

Variables Scope

❖Variables declared with ***var*** keyword can be redeclared at any point in the code even within the same execution context. This is not the case for variables defined with ***let*** and ***const*** keywords as they can only be declared once within their lexical scope

❖Variables declared with ***const*** even though its value cannot be reassigned to, it is still mutable

Scope.html

©2019 Clayfin

Hoisting

- ❖The primary importance of hoisting is that it allows you to use functions before you declare them in your code
- ❖ variable and function declarations are put into memory during the compilation phase but stay exactly where you typed them in your code
- ❖ all variables defined with the ***var*** keyword have an initial value of ***undefined***.
- ❖ variables defined with ***let*** or ***const*** keywords when hoisted are in a state called the **Temporal Dead Zone** and are not initialized until their definitions are evaluated

Hoisting.html

Method Chaining

Chaining Methods, also known as Cascading, refers to repeatedly calling one **method** after another on an object, in one continuous line of code

```
Object . Method1 ( ) . Method2 ( ) . Method3 ( );
```

Chaining Methods can be done only if all the methods are Chainable by returning the current object

MethodChaning.html

Important Concepts

Clayfin

Template Strings
console.log(`My name is ${name}  and aged ${age}`);

Difference between == and === operator
(equality of value and same data type)

Computed Property Names - allows the names of object properties to be determined dynamically, i.e. computed

ComputedPropertyNames.html

DeStructuring assignment - to unpack values from arrays, or properties from objects, into distinct variables

Destructuring.html

setTimeout and setInterval

setTimeout() method calls a function or evaluates an expression after a specified number of milliseconds.
The function is only executed once.
If you need to repeat execution, use the **setInterval**() method.
Use **clearTimeout**() method to prevent the function from running

TimerId = setTimeout(*function, milliseconds, param1, param2, ...*)

TimerId = setInterval(function, *milliseconds, param1, param2, ...*)

clearTimeout(TimerId)

SetTimeout.html

Classes in ES6

Javascript did not have classes till ES6.
It used Prototype based inheritance for OOPs

In ES6, Class keyword was introduced which was more of Syntactical Sugaring as the internal concept and implementation remained the same as ES5

Subclass can be created using extend keyword

Class.html

Encapsulation

Clayfin

Encapsulation can be done in javascript by the use of var instead of this for the properties and methods in the object

Encapsulation.html

Spread Operator {...}

Spread operator {...} allows an iterable such as an array expression or string to be expanded in places where zero or more arguments (for function calls) or elements (for array literals) are expected, or an object expression to be expanded in places where zero or more key-value pairs (for object literals) are expected

```
var myfunc = function ( ...n ) {
        console.log(n);
};
myfunc(1,2,3);
myfunc(1,2,3,4,5);
```

Spread.html

Functions As Arguments and Return Value

Functions can be passed as Arguments for another Functions

Functions can return Functions as a return Value

FunctionsAsArgandReturn.html

©2019 Clayfin

Arrow Functions

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

ES5

```
let  sum  = ( a  ,  b ) =>  (a + b) ;
```

ES6

Does not have it own “this” and uses Parent “this”
Cleaner alternative for that = this pattern

Arrow.html

Call – Apply - Bind

Clayfin

call, apply, and bind methods allows to borrow methods from other objects (can enable multiple inheritance)

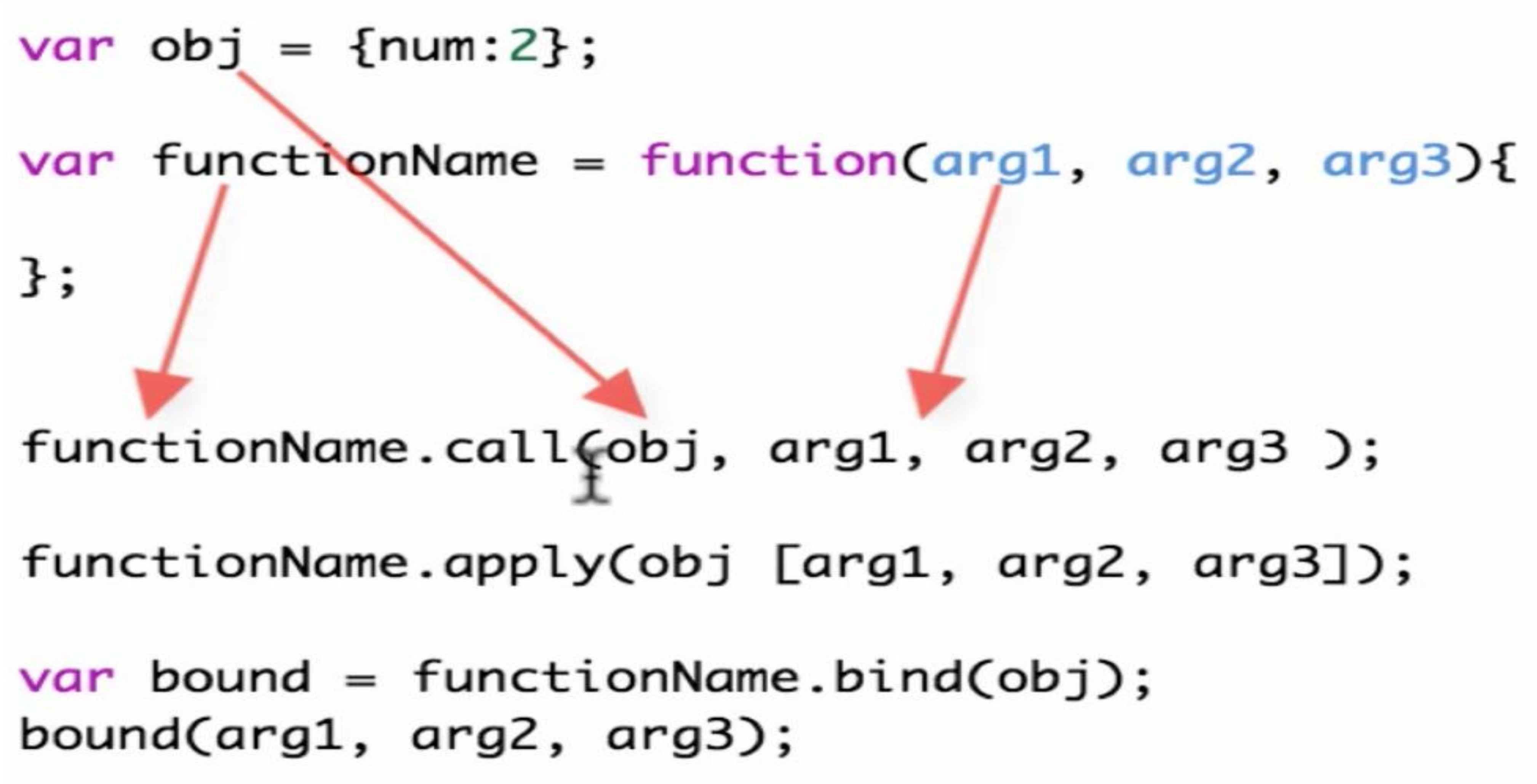
call, apply, and bind methods to set the this keyword independent of how the function is called. This is especially useful for the callbacks

call and apply call the function immediately
Time of binding immediate and Time of Execution immediate

bind returns a function that, when later executed
Time of binding immediate and Time of Execution Future

©2019 Clayfin

Call – Apply - Bind



```
var obj = {num:2};

var functionName = function(arg1, arg2, arg3){

};


functionName.call(obj, arg1, arg2, arg3 );

functionName.apply(obj [arg1, arg2, arg3]);

var bound = functionName.bind(obj);
bound(arg1, arg2, arg3);
```

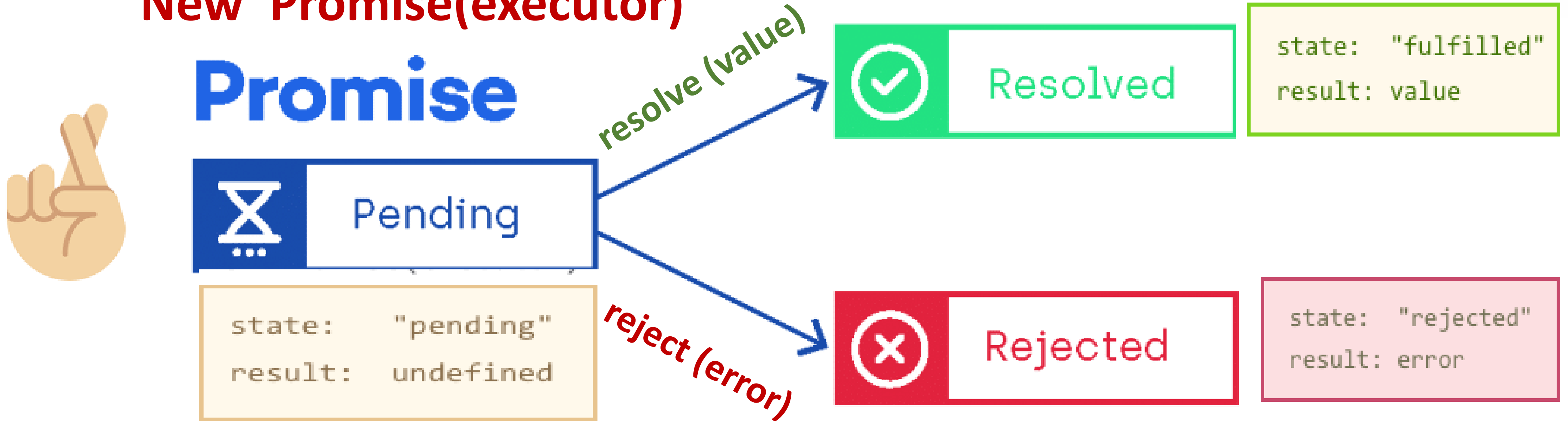
Call-Apply-Bind.html

Promises

A Promise object represents a value that may not be available yet, but will be resolved or rejected at some point in the future

Promises are used to handle asynchronous operations in JavaScript. They are easy to manage when dealing with multiple asynchronous operations where callbacks can create callback hell leading to unmanageable code

New Promise(executor)



Pending: not yet fulfilled or rejected
Fulfilled: Promise resolved successfully
Rejected: Promise rejected

Promise.html

Promises

```
const promise = new Promise((resolve, reject) => {
  const request = new XMLHttpRequest();

  request.open('GET', 'https://api.clayfin.com/users');
  request.onload = () => {
    if (request.status === 200) {
      resolve(request.response); // we got data here, so resolve the Promise
    } else {
      reject(Error(request.statusText)); // status is not 200 OK, so reject
    }
  };
  request.onerror = () => {
    reject(Error('Error fetching data.')); // error occurred, reject the  Promise
  };
  request.send(); // send the request
});

console.log('Asynchronous request made.');

promise.then((data) => {
  console.log('Got data! Promise fulfilled.');
  console.log(JSON.parse(data));
}, (error) => {
  console.log('Promise rejected.');
  console.log(error.message);
});
```

Modules in Javascript : AMD , CommonJS , ES6Modules

ES6 Modules

export keyword is used when we want to make something available somewhere

Named Exports

- Variable
- Class
- Function

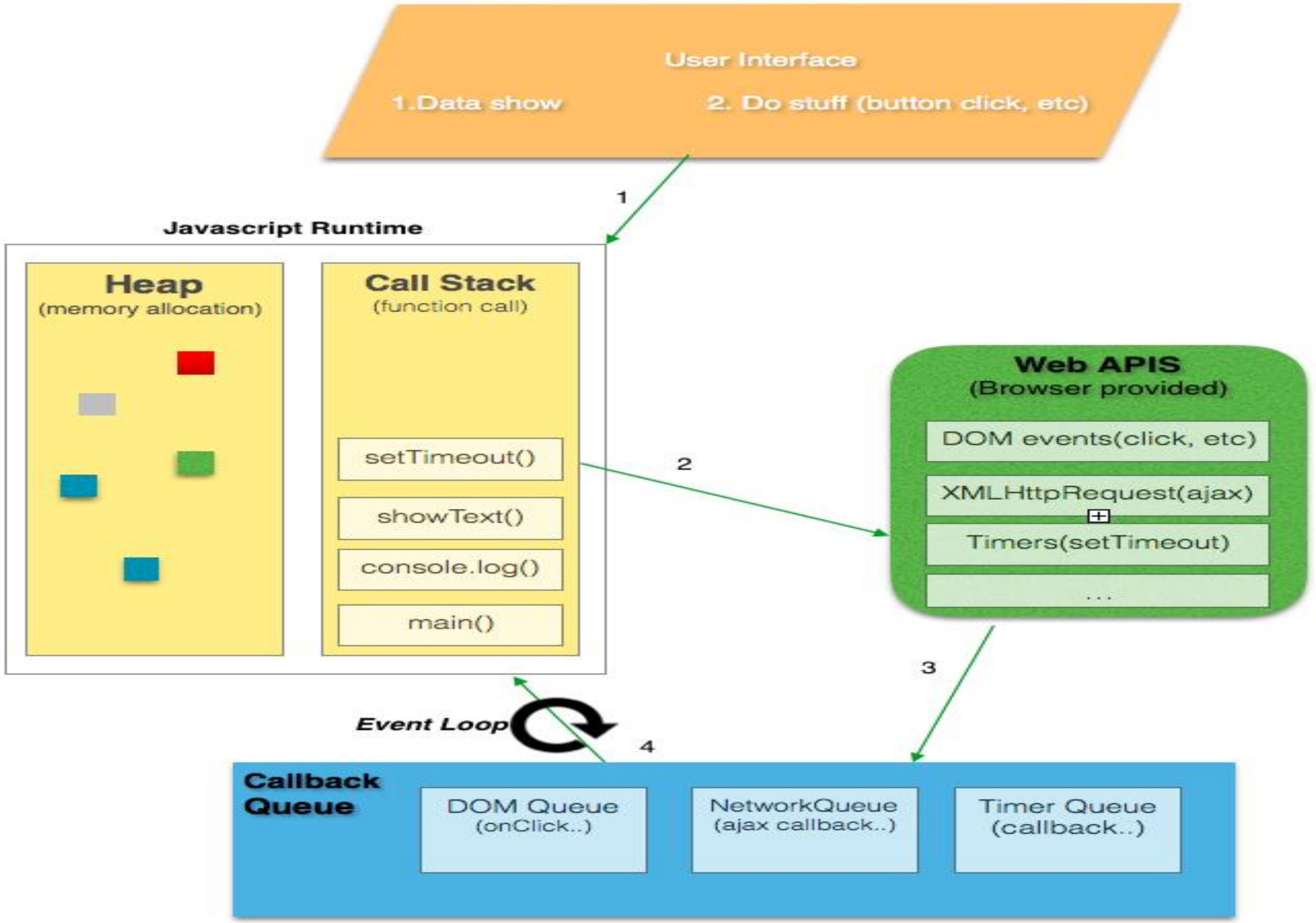
Default export - need to export only a single value

import keyword is used to access what export has made available

ES6Modules.html

Execution Model

Clayfin



Thank You

www.clayfin.com