## Why put the script tag in the last of HTML file?

Putting the script tag in the last can reduce the delay time, users won't wait a long time to open the webpage. Because the render direction is from top to bottom.

## How many data types does JavaScript have?

7 Primitives types: Number String Boolean Null Undefined (Symbol bigint: ES6)
+ Array, Function +Object
///
console.log(0.1 + 0.2 === 0.3);
What is output? Why?
///
**value type vs reference type**

value type: Boolean, String, Number, null, undefined.

reference type: Array, Function. Object.

What happen when assign reference type and assign value type
What happen passing reference type and value type as function arguments
How to compare between value types and reference types.

const with obj/array( you can still modify the obj/array because of reference type)

///

```
function App1()
{
  return {
     bar: "hello"
  };
}

function App2()
{
  return {
     bar: "hello"
  };
}
console.log(App1() === App2()) //false
console.log(App1().bar === App2().bar) // true
```

What is the output? And why?
///

**Shallow clone vs deep clone**
Shallow clone: Object.assign, spread operator
Deep clone: use lodash, or JSON.parse(JSON.stringify(obj)) (limited) or structuredClone (browser support limitated)

**var vs let vs const**
var: **variable hoisting** ,function scope (function keyword also has hoisting), can redeclare
let/const: no variable hoisting, block scope, cannot redeclare

```
var variable = 10;
(()=>{
  variable_3 = 35;
  console.log(variable_3);
  var variable_3 = 45;
  variable_2 = 15;
  console.log(variable);
})();
console.log(variable_2)
console.log(variable_3)
```

**IIFE (Immediately Invoked Function Expression)**
function(i) {
}
(function(i) {
})(i)
issue:

```
for(var i = 0;i < 10; i++){
   setTimeout(function(){
       console.log(i);
   }, 1000);
}
```

Solution:

```
for(let i = 0;i < 10; i++){
   setTimeout(function(){
       console.log(i);
   }, 1000);
}
```

```
for(var i = 0; i < 10; i++){
    (function(i){
        setTimeout(function(){
            console.log( i);
        }, 1000);
    })(i);
}
```

**Closure**
Simple: a function returns another function. The returned function can access the inner scope of the first function.
Full: A closure is an inner function that has access to the outer function's variables. The closure has three scope chains: it has access to its own scope, it has access to the outer function's variables, and it has access to global variables.
///
let addNum= createBase(6);
addNum(10); // output: 16
addNum(21); // output: 27
Write a code for achieving above requirements
///

**How async works and why we need async**
**Event loop.** We want non-blocking and avoid DOM render conflicting
https://www.youtube.com/watch?v=8aGhZQkoFbQ&vl=en

**What is Promise. How async / await is different from Promise.**
Promise is like a container. In this container have the result of future things.
Promise is a cleaner way for running asynchronous tasks. **Avoid callback hell**.
it has two states, **pending** & **settled**. And **Settled** has **fulfilled** & **rejected**
Promise use methods like "**.then**" and "**catch**" to deal with all kinds of asynchronous operation

Async/await is the same thing as Promise. It is just syntax sugar that allows us to write async code like sync code.

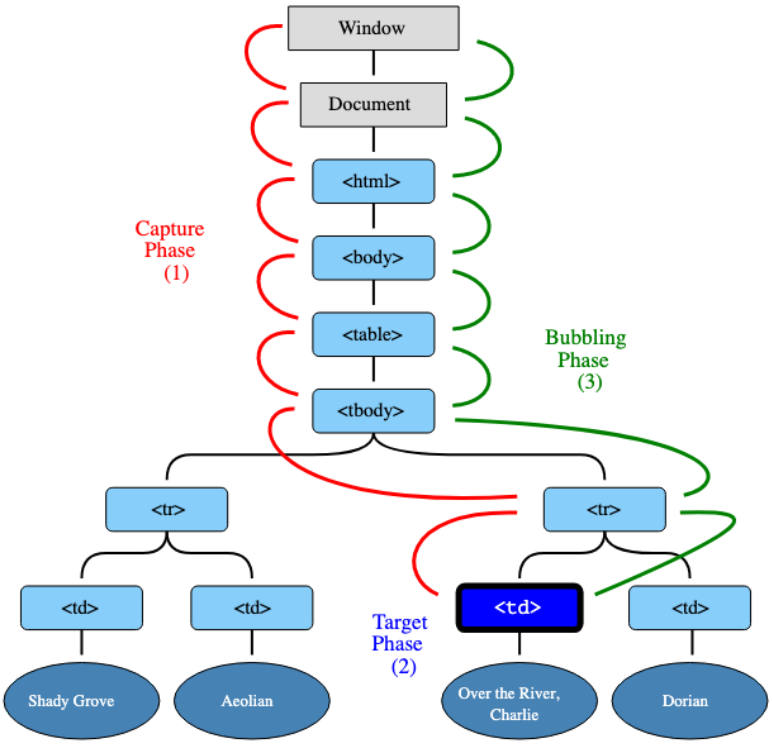
Promise.all().then() Promise.race() then Promise.allSettled()
Promise.resolve(), Promise.reject()

priority of Promise.resolve() vs setTimeout(callback, 0);
///
```
let promise = new Promise((resolve, reject) => {
  let workout = true
  if (workout) {
```

```
    resolve('good')
  } else {
    reject('bad')
  }
})

console.log(1)
setTimeout(() => console.log(2), 0)
console.log(3)
promise.then(res => console.log(res))
What is the output order? Why?
// micro task vs macro task
///
```

**What is event bubbling, event capturing.**

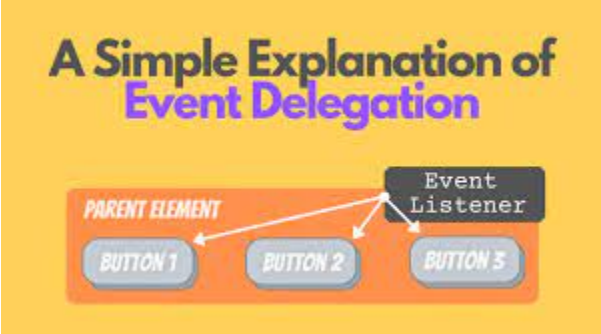



**e.target VS e.currentTarget ?**

e.target: The thing under the mouse (the thing that triggers the event).
e.currentTarget is the  element with the added event listener.

## what's Event Delegation?

Event Delegation is the method. this method will let you create one event listener to trigger all child elements. Each child element will be targeted by a unique ID.




**How “this” works in Javascript**
“this" means the environment object where the function is running.
**With “Function” keyword:**
“this” in constructor function. -> the function called with “new” -> this refers the instance
“this” in an object method function -> refers to who calls the function (the thing before the “.”)
“this” in a plain function -> window(global object in Nodejs) - not in strict mode
“this” in event handler -> the element fires the event
**Arrow function**
“this” points to closest “this” when create the function, it is lexical scope
Arrow function cannot be the constructor function
**How “bind”, “apply”, “call” works**
The purpose is to change the “this” referring.

```
//bind
const newFunc = func.bind(antherObj);
newFunc(arg1, arg2)

func.apply(anotherObj, [arg1, arg2])

func.call(anotherObj, arg1, arg2)

///
var test = 1
let obj = {
  test: 'test',
  method: function() {
```

```
    console.log(this.test)
  }
}

let app = obj.method
app()
```

What is output? And why?

///

**Cross-Origin Resource Sharing**

web browsers restrict cross-origin requests due to the Same-Origin Policy. This policy prevents scripts running on one origin (domain, protocol, and port) from accessing resources on a different origin.

To enable cross-origin requests, the server needs to respond with specific HTTP headers

- Access-Control-Allow-Origin: *
- Access-Control-Allow-Methods
- Access-Control-Allow-Headers
- Access-Control-Allow-Credentials

Simple requests(Get) are sent directly to the server, and the browser checks the response headers for CORS permissions. Preflighted requests are sent when the request is considered complex,  and the browser first sends an HTTP OPTIONS preflight request to check if the server allows the actual request.

Resolving CORS:

- Server-Side Configuration
- Proxy Server
- JSONP

**Javascript Prototype**

1. Why do we need the “prototype”? To re-use method in the object
2. Each **function** automatically has a “**prototype**” property. The value of “**prototype**” is an object.

3. Each **object(instance)** has a “**__proto__**” property. The “**__proto__**” refers the “**prototype**” of its creator(the **function**).

4. You can use a **function**(constructor) with “new” keyword to create an object(instance). Then the created **object’s** “**__proto__**” refers the “**prototype**” of the

function(constructor)

5. the **object** will have a “**constructor**” property automatically, which refers to the constructor(the function)

6. Arrow function cannot be a constructor

7. **6**

```
function MyFunc() {
  this.name = "name";
}
const instance = new MyFunc();
console.log(instance.__proto__ === MyFunc.prototype); // true
console.log(MyFunc === MyFunc.prototype.constructor); // true
```

**ES6 “class” syntax is just a syntax sugar of prototype**

///

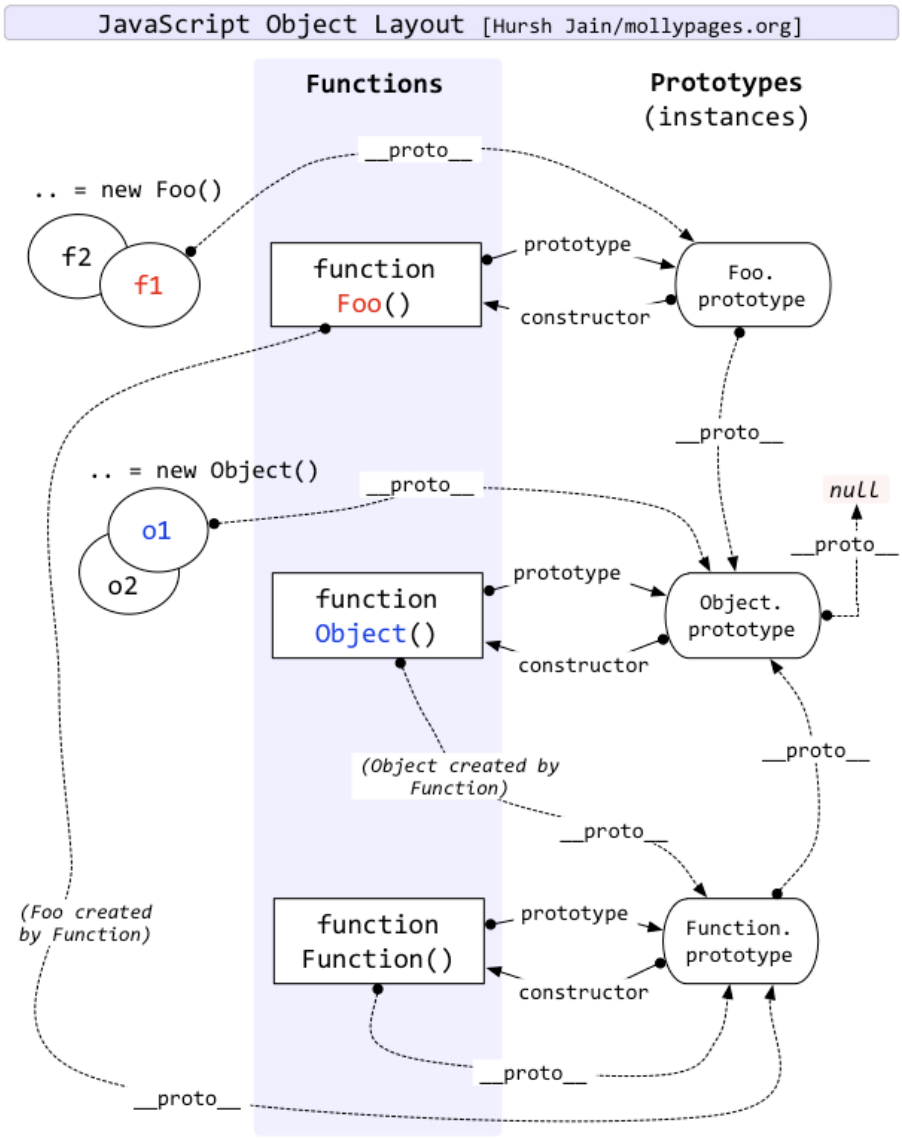
implemente

[1,2,3,4,5].print(); // “1,2,3,4,5”

[2,3,4,5,6].print(); // “2,3,4,5,6”

```
Array.prototype.print = function() {
        console.log(this.join(‘,’));
}
```

///


JavaScript Object Layout [Hursh Jain/mollypages.org]
Functions
Prototypes
(instances)
.. = new Foo()
f2
f1
__proto__
function
Foo()
prototype
constructor
Foo.
prototype
__proto__
.. = new Object()
o1
o2
__proto__
null
__proto__
function
Object()
prototype
constructor
Object.
prototype
(Object created by
Function)
__proto__
__proto__
(Foo created
by Function)
function
Function()
prototype
constructor
Function.
prototype
__proto__
__proto__