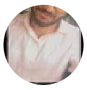


Himanshu Gupta
Posted on 1 mars 2023

💖 4

JavaScript Console Tips. Other console tools in JavaScript

#javascript #console #consolelog #debugg

1.console.log()
Let's imagine we have a variable called name and we want to log it in the console.

Often we find ourselves writing this:

```
console.log('name', name)
```

Since ES2015, we can use object shorthand notation whenever we want to log out things like this. That means we can write the following:

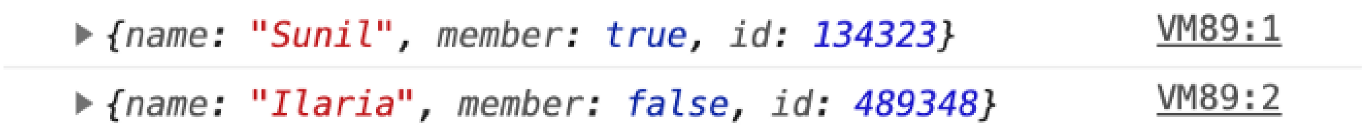
```
console.log({name})
```

And this will output the same thing.

2. A better way to log multiple items
Imagine a scenario when you have a bunch of objects you need to log into the console.

```
const sunil = { name: "Sunil", member: true, id: 134323 };
const ilaria = { name: "Ilaria", member: false, id: 489348};
If we were to console.log() these out individually, like so:

console.log(sunil)
console.log(ilaria)
```

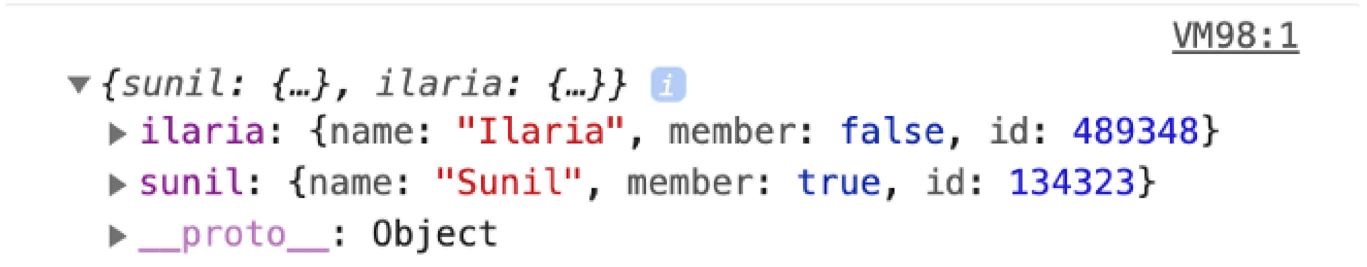
It would be slow, tedious, and would not display the variable names alongside the logged data



```
▶ {name: "Sunil", member: true, id: 134323}        VM89:1
▶ {name: "Ilaria", member: false, id: 489348}      VM89:2
```

Instead, we can use our good friend from the first example!

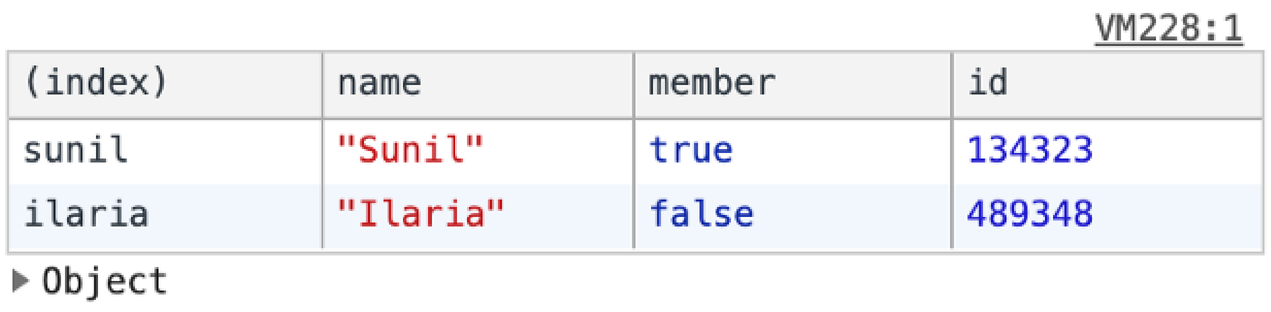
console.log({sunil, ilaria})
And this will give us a better formatted, quicker solution to console.logging multiple items, and will display the variable names alongside each!



```
                                                    VM98:1
▼ {sunil: {…}, ilaria: {…}} i
  ▶ ilaria: {name: "Ilaria", member: false, id: 489348}
  ▶ sunil: {name: "Sunil", member: true, id: 134323}
  ▶ __proto__: Object
```

3. Why use lines when you can use tables?

We can take this a step further by putting all of these together in a table to make it more readable. Whenever you have objects with common properties or an array of objects use console.table() . Here we can use console.table({ foo, bar}) and the console shows:



VM228:1

(index)	name	member	id
sunil	"Sunil"	true	134323
ilaria	"Ilaria"	false	489348

▶ Object

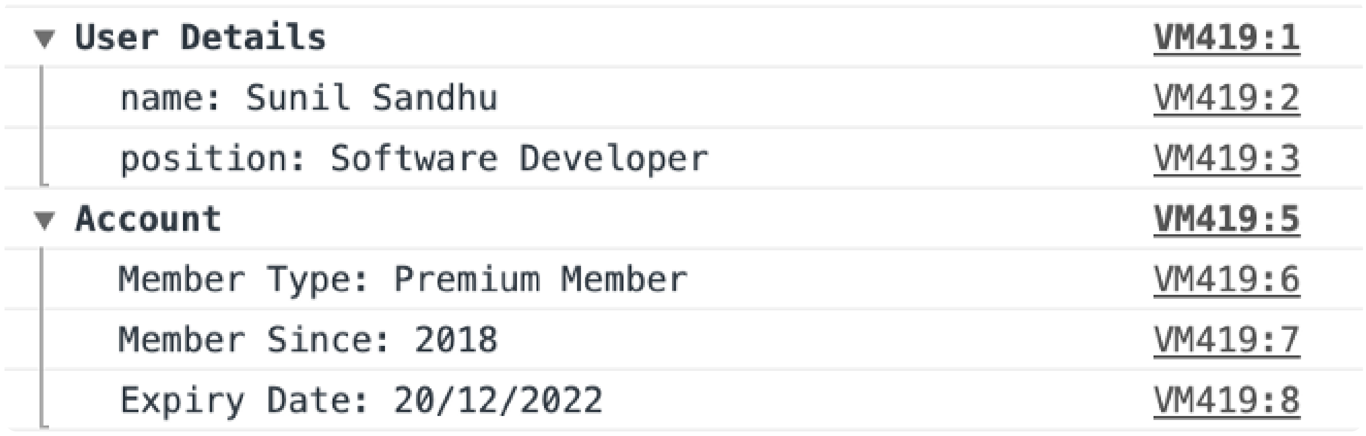
4. Group grouped logs

This can be useful for those moments when you are logging a lot of different things and want to group or nest relevant details together.

Maybe you are logging information in a few different functions and you want a way to clearly label which function the information is coming from.

For example, if you're logging a user's details:

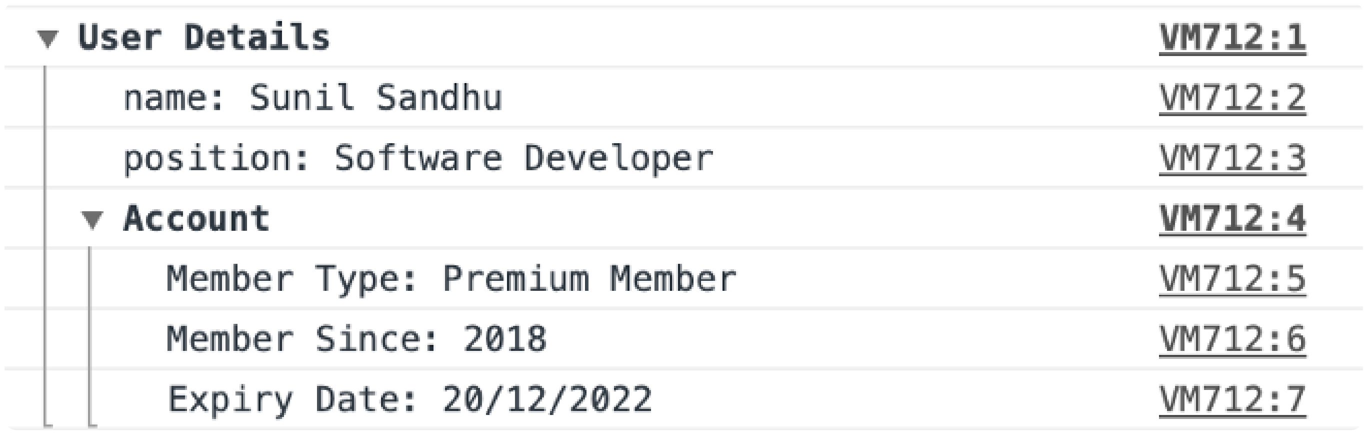
```
console.group('User Details');
console.log('name: Sunil Sandhu');
console.log('position: Software Developer');
console.groupEnd();console.group('Account');
console.log('Member Type: Premium Member');
console.log('Member Since: 2018');
console.log('Expiry Date: 20/12/2022');
console.groupEnd();
```



```
▼ User Details                          VM419:1
    name: Sunil Sandhu                  VM419:2
    position: Software Developer        VM419:3
▼ Account                               VM419:5
    Member Type: Premium Member         VM419:6
    Member Since: 2018                  VM419:7
    Expiry Date: 20/12/2022             VM419:8
```

You can even nest groups inside of others if you want:

```
console.group('User Details');
console.log('name: Sunil Sandhu');
console.log('position: Software Developer');console.group('Account');
console.log('Member Type: Premium Member');
console.log('Member Since: 2018');
console.log('Expiry Date: 20/12/2022');
console.groupEnd();
console.groupEnd();// notice that we have two groupEnd() calls at the end as we wan
```

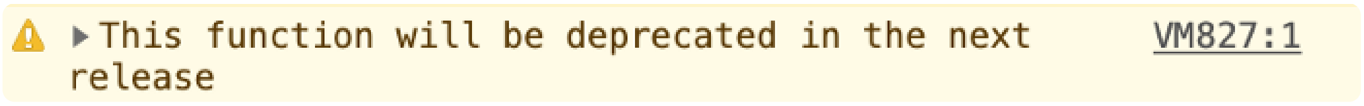


```
▼ User Details                          VM712:1
    name: Sunil Sandhu                  VM712:2
    position: Software Developer        VM712:3
  ▼ Account                             VM712:4
      Member Type: Premium Member       VM712:5
      Member Since: 2018                VM712:6
      Expiry Date: 20/12/2022           VM712:7
```

5. Better warnings

Want to increase the visibility of certain bits of information being logged? console.warn() will display information with a yellow background:

```
console.warn('This function will be deprecated in the next release')
```

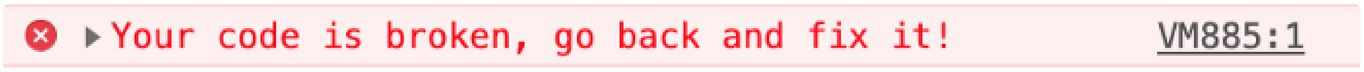


```
⚠ ▶This function will be deprecated in the next        VM827:1
  release
```

6. Better error logging

Maybe you want to take things one step further and use the same type of logging you get whenever you receive those scary red console logs. You can achieve that like so:

```
console.error('Your code is broken, go back and fix it!')
```



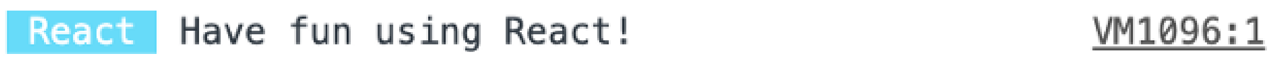
```
✖ ▶Your code is broken, go back and fix it!        VM885:1
```

7. Custom console styling

Take your CSS skills and start using them in the console!

You can use a %c directive to add styling to any log statement.

```
console.log('%c React ',
            'color: white; background-color: #61dbfb',
            'Have fun using React!');
```

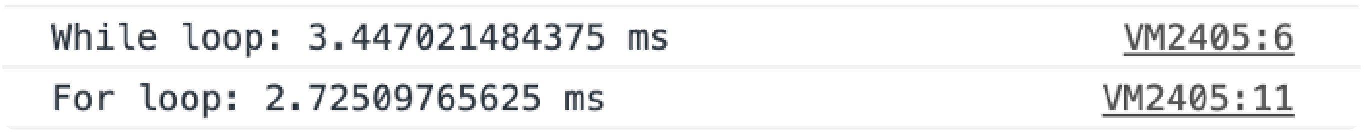


```
 React  Have fun using React!        VM1096:1
```

8. Time the speed of your functions

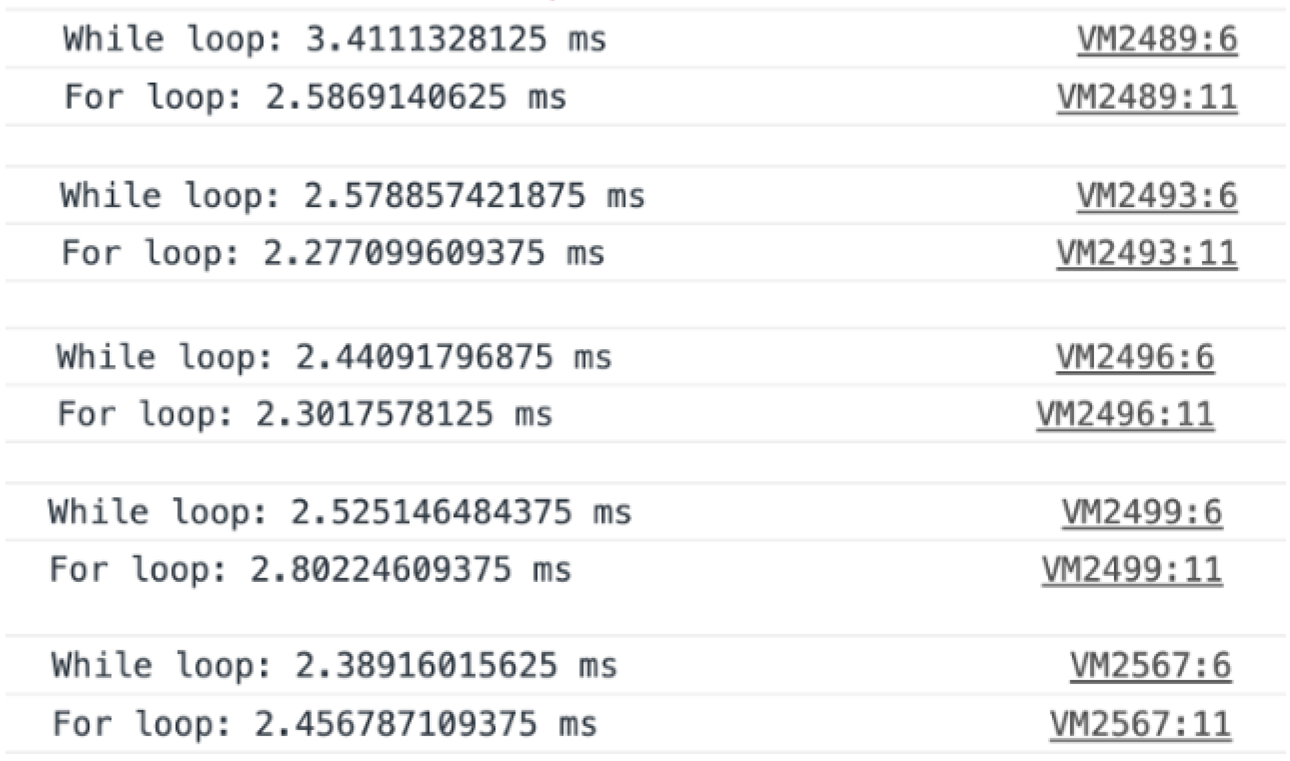
Ever been curious to know how fast a particular function runs? You can write something like this:

```
let i = 0;
console.time("While loop");
while (i < 100000) {
  i++;
}
console.timeEnd("While loop");console.time("For loop");
for (i = 0; i < 100000; i++) {
  // For Loop
}
console.timeEnd("For loop");
```



```
While loop: 3.447021484375 ms        VM2405:6
For loop: 2.72509765625 ms           VM2405:11
```

It's worth bearing in mind that the speeds you see here are not static. In other words, it can give you an idea of which is faster between the two at that exact moment in time, but other factors influence this, such as the computer it is running on, other things being executed at the time etc. In fact, if we run this 5 more times, we will see different results for each:

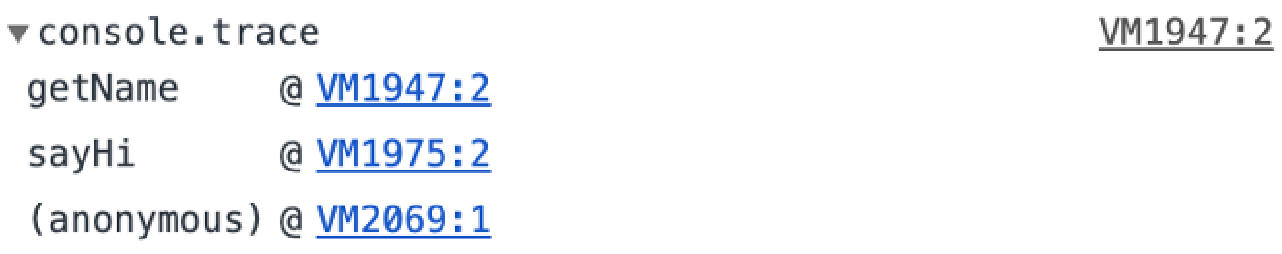


```
While loop: 3.4111328125 ms                VM2489:6
For loop: 2.5869140625 ms                  VM2489:11

While loop: 2.578857421875 ms              VM2493:6
For loop: 2.277099609375 ms                VM2493:11

While loop: 2.44091796875 ms               VM2496:6
For loop: 2.3017578125 ms                  VM2496:11

While loop: 2.525146484375 ms              VM2499:6
For loop: 2.80224609375 ms                 VM2499:11

While loop: 2.38916015625 ms               VM2567:6
For loop: 2.456787109375 ms                VM2567:11
```

9. Better stack traces

console.trace() outputs a stack trace to the console and displays how the code ended up at a certain point. This can really come in handy when trying to debug complex code that might be making calls in lots of different places. While the following is not an example of "complex code", it'll at least explain how this works:

```
const sunil = {name: "Sunil", member: true, id: 134323};function getName(person) {
    console.trace()
    return person.name;
}function sayHi(person) {
    let _name = getName(person)
    return `Hi ${_name}`
}
The console.trace() would return this:
```



```
▼console.trace                              VM1947:2
  getName      @ VM1947:2
  sayHi        @ VM1975:2
  (anonymous)  @ VM2069:1
```

We can click on those blue links and it will take us to the moment where that console.trace() was made in our code.

10. console.trace()

console.trace() works the exact same as console.log(), but it also outputs the entire stack trace so you know exactly what's going on.

```
const outer = () => {
  const inner = () => console.trace('Hello');
  inner();
};
outer();
/*
  Hello
  inner @ VM207:3
  outer @ VM207:5
  (anonymous) @ VM228:1
*/
```

11. Logging levels

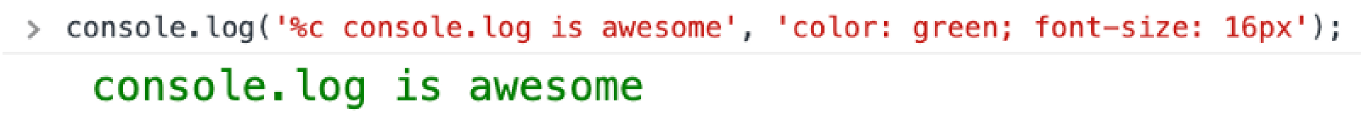
There are a few more logging levels apart from console.log(), such as console.debug(), console.info(), console.warn() and console.error().

```
console.debug('Debug message');
console.info('Useful information');
console.warn('This is a warning');
console.error('Something went wrong!');
```

12. Formatting

The console.log() method also allows you to add some styling to your messages. To do that you need to add a %c specifier to your message and pass CSS styling as an additional parameter to the log() method.

```
console.log('%c console.log is awesome', 'color: green; font-size: 16px');
```



```
> console.log('%c console.log is awesome', 'color: green; font-size: 16px');
   console.log is awesome
```

We can click on those blue links and it will take us to the moment where that console.trace() was made in our code.

10. console.trace()

console.trace() works the exact same as console.log(), but it also outputs the entire stack trace so you know exactly what's going on.

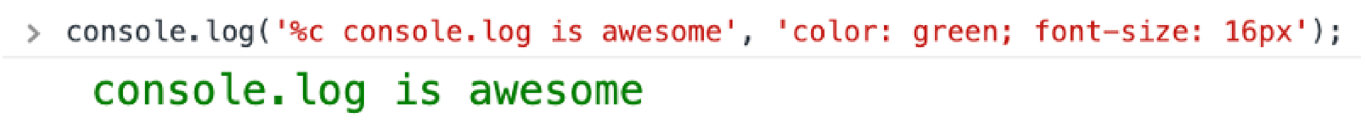
```
const outer = () => {
  const inner = () => console.trace('Hello');
  inner();
};
outer();
/*
  Hello
  inner @ VM207:3
  outer @ VM207:5
  (anonymous) @ VM228:1
*/```



**11. Logging levels**
There are a few more logging levels apart from console.log(), such as console.debug
```

console.debug('Debug message');
console.info('Useful information');
console.warn('This is a warning');
console.error('Something went wrong!');```

12. Formatting

The console.log() method also allows you to add some styling to your messages. To do that you need to add a %c specifier to your message and pass CSS styling as an additional parameter to the log() method.

console.log('%c console.log is awesome', 'color: green; font-size: 16px');

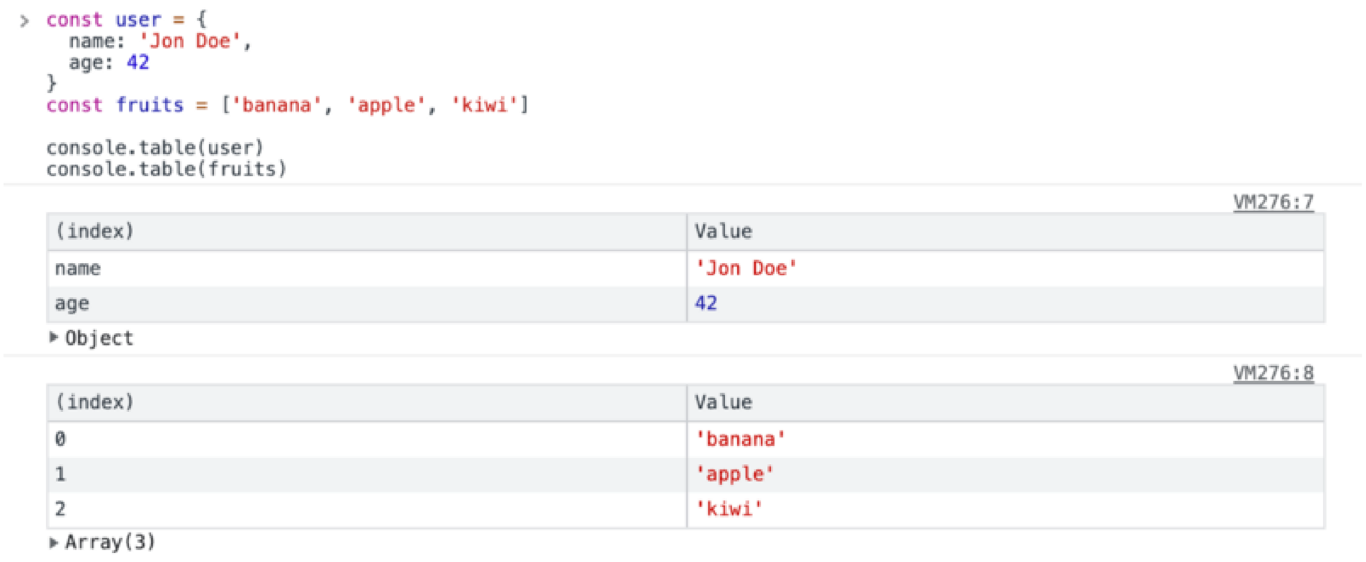


```
> console.log('%c console.log is awesome', 'color: green; font-size: 16px');
   console.log is awesome
```

13. console.table

Use console.table() to display object or array data as a table.

```
const user = {
  name: 'Jon Doe',
  age: 42
}
const fruits = ['banana', 'apple', 'kiwi']
console.table(user)
console.table(fruits)
```



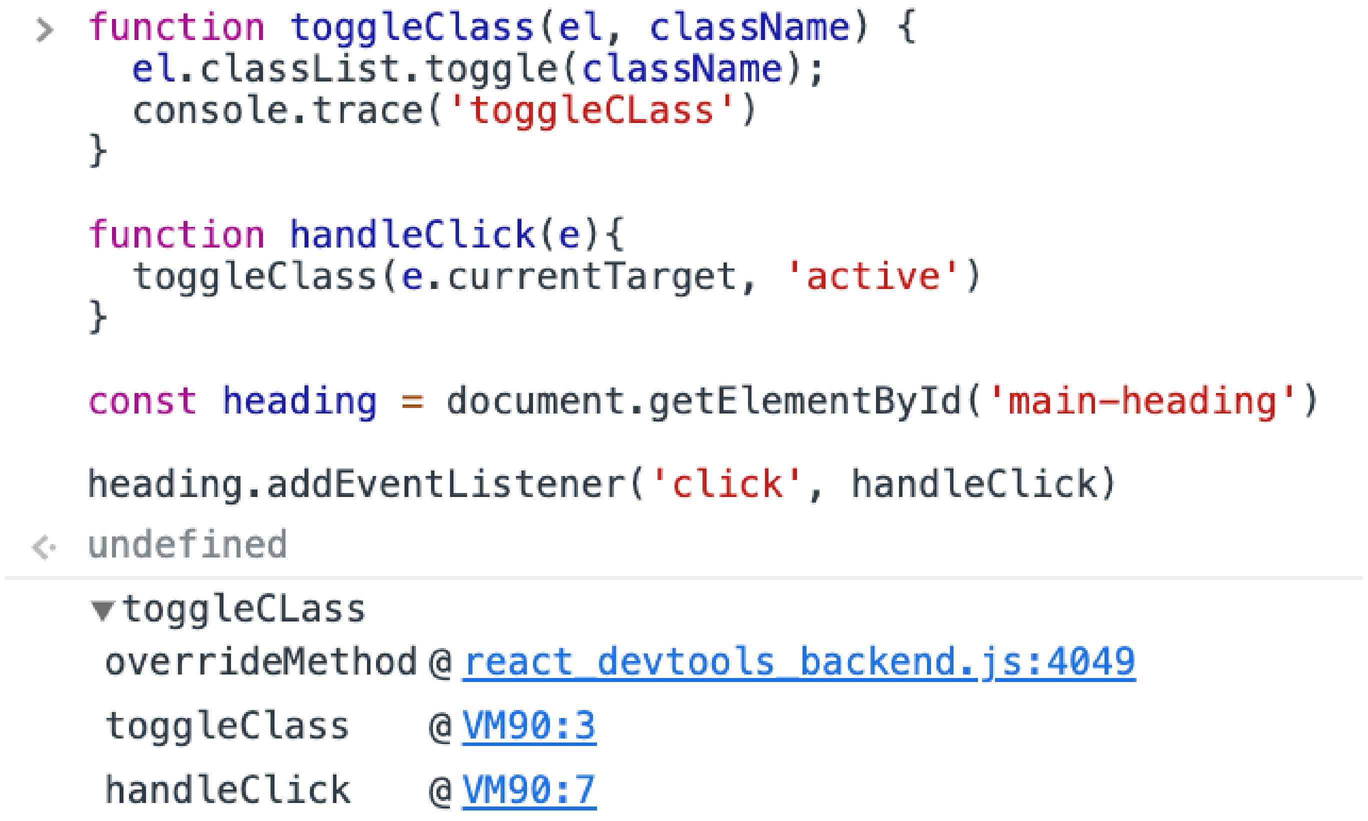
```
> const user = {
    name: 'Jon Doe',
    age: 42
  }
  const fruits = ['banana', 'apple', 'kiwi']

  console.table(user)
  console.table(fruits)
```

```
                                                                     VM276:7
(index)                                 Value
name                                    'Jon Doe'
age                                     42
▶ Object
                                                                     VM276:8
(index)                                 Value
0                                       'banana'
1                                       'apple'
2                                       'kiwi'
▶ Array(3)
```

14. console.trace

A handy method to log to console the stack trace. It will show you the function call path up until your console.trace() call. Quite useful during the debug process to instantly show which functions gets called.

```
function toggleClass(el, className) {
  el.classList.toggle(className);
  console.trace('toggleCLass')
}
function handleClick(e){
  toggleClass(e.currentTarget, 'active')
}
const heading = document.getElementById('main-heading')
heading.addEventListener('click', handleClick)
```



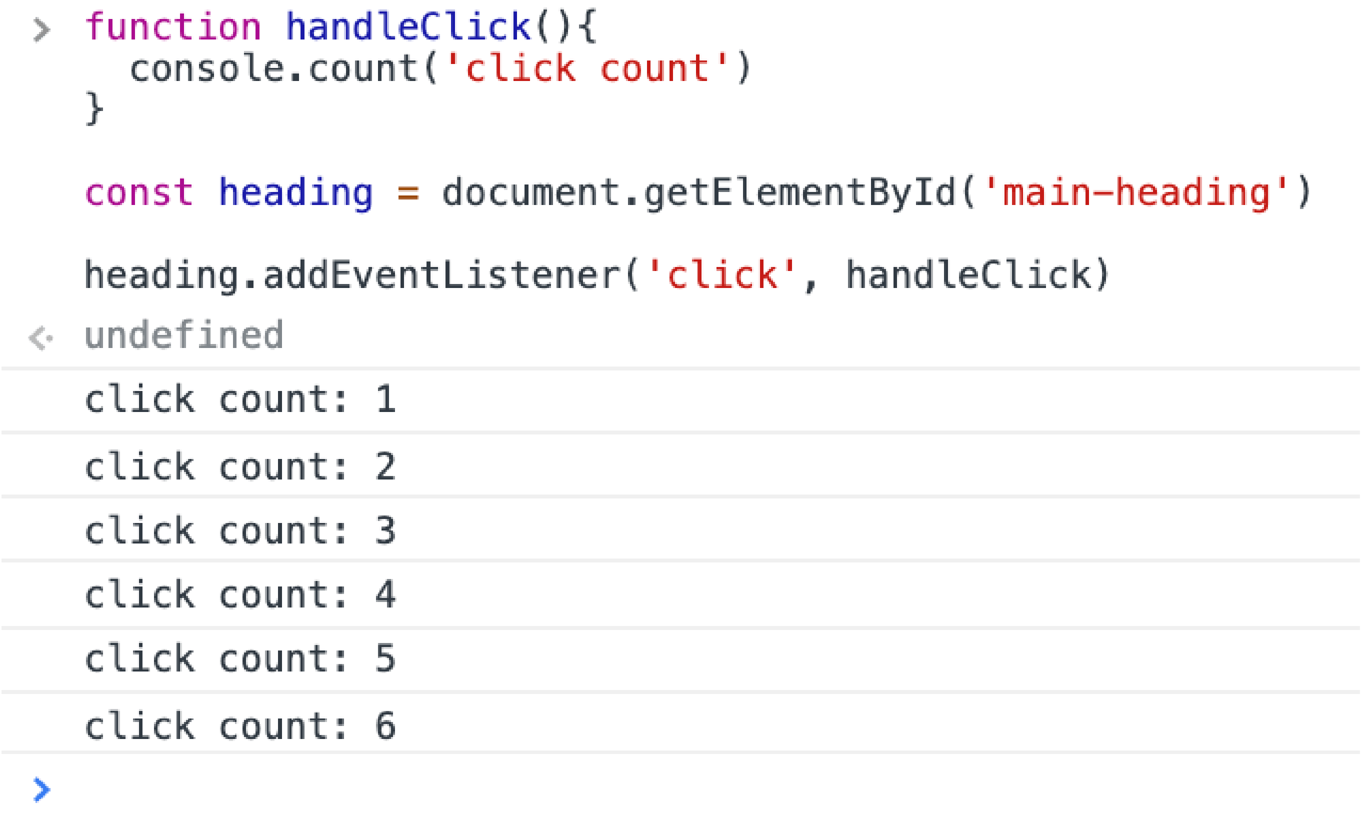
```
> function toggleClass(el, className) {
    el.classList.toggle(className);
    console.trace('toggleCLass')
  }

  function handleClick(e){
    toggleClass(e.currentTarget, 'active')
  }

  const heading = document.getElementById('main-heading')

  heading.addEventListener('click', handleClick)
< undefined
  ▼toggleCLass
   overrideMethod @ react_devtools_backend.js:4049
   toggleClass    @ VM90:3
   handleClick    @ VM90:7
```

15. console.count

A method acts as a counter and is used to output the number of times the particular count() method has been called.

You can pass a string (label) as an argument and it will output like a regular console.log with a number next to it. If no argument is specified then the default label will be shown.

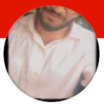
```
function handleClick(){
  console.count('click count')
}
const heading = document.getElementById('main-heading')
heading.addEventListener('click', handleClick)
```



```
> function handleClick(){
    console.count('click count')
  }

  const heading = document.getElementById('main-heading')

  heading.addEventListener('click', handleClick)
<· undefined
  click count: 1
  click count: 2
  click count: 3
  click count: 4
  click count: 5
  click count: 6
>
```

Top comments (0)

Code of Conduct • Report abuse



Himanshu Gupta

I have no special talent. I am only passionately curious.

LOCATION
Chandigarh

EDUCATION
MCA

WORK
Dev

JOINED
27 déc. 2019

More from Himanshu Gupta

What is Event propagation, capturing, bubbling ?

#javascript #beginners #tutorial

What is the Difference between Spread and Rest Operator in JavaScript

#javascript #beginners #tutorial #typescript

What is Javascript Singleton Object with example

#javascript #development #softwaredevelopment #developer