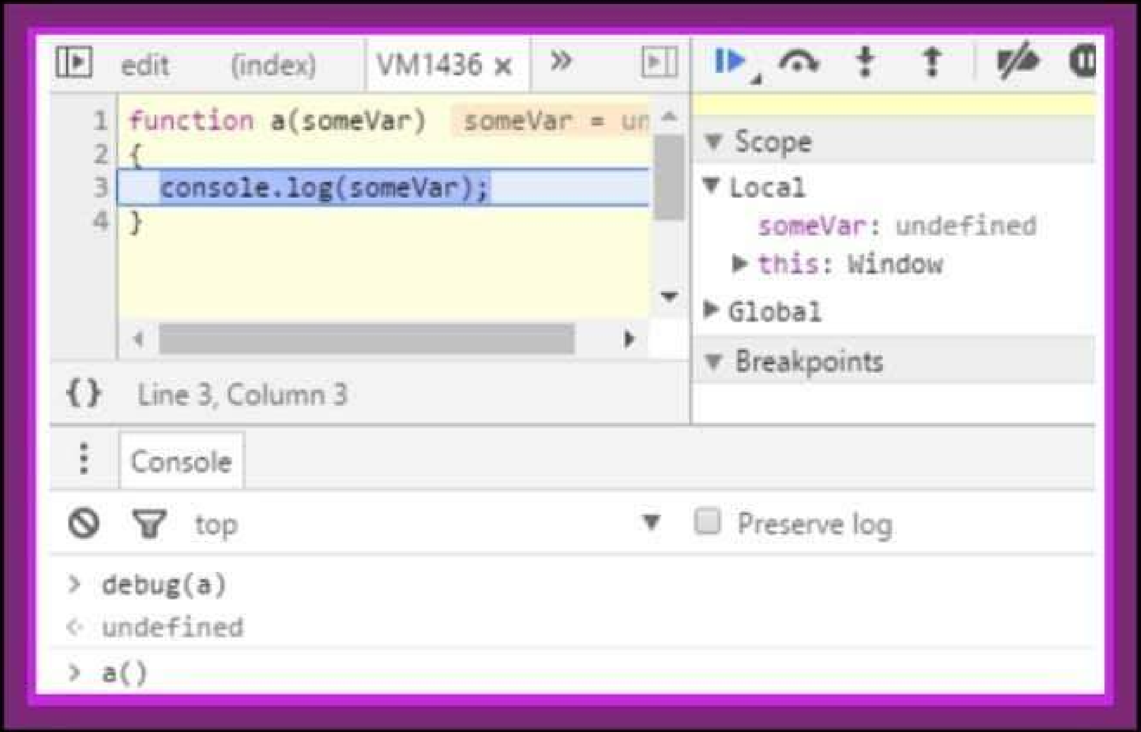
7 METHODS TO DEBUG JAVASCRIPT CODE

ALERT

```
alert('Hi')
```

Alert method is used from beginning of JavaScript development. It displays a value of string or variable into an alert box. Not much useful currently as more advanced methods available. It's annoying to get a dialog on the page every time when line getting executes.

DEBUG



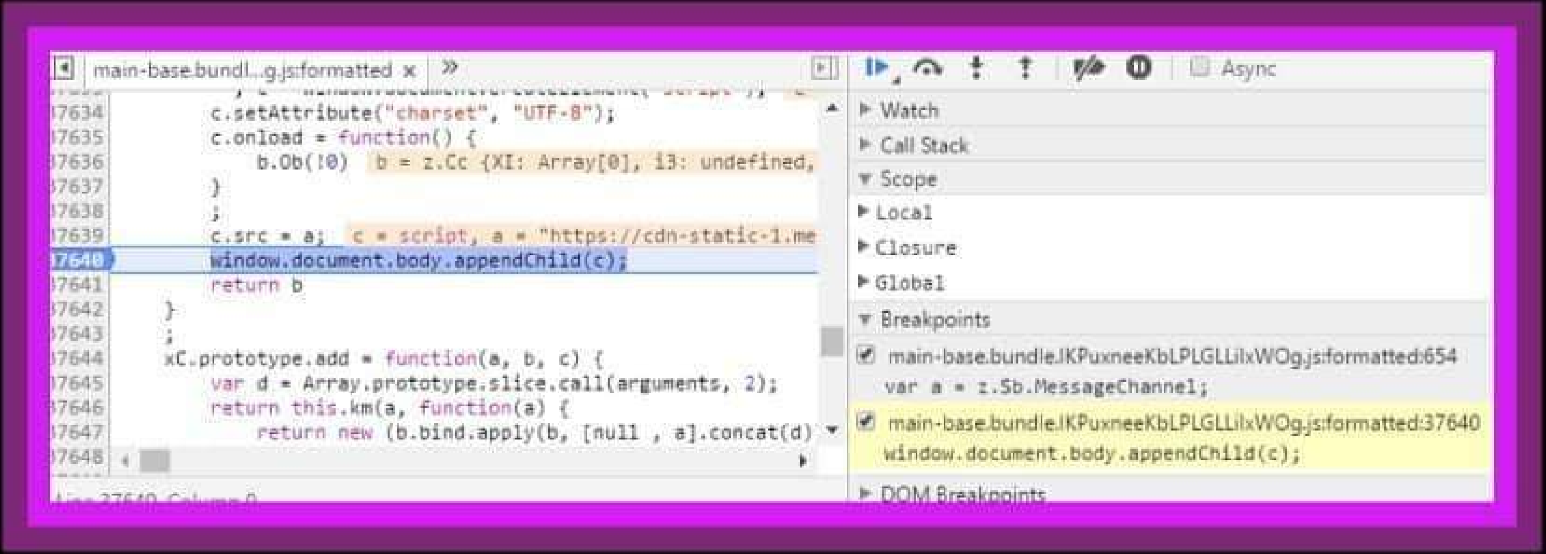
```
function a(someVar)  someVar = un
{
  console.log(someVar);
}
```

```
▼ Local
    someVar: undefined
  ▶ this: Window
▶ Global
```

```
> debug(a)
<· undefined
> a()
```

Sometimes you want to debug on some function call and search result of function name search returns nothing may be because of minification or special chars. For such scenario debug method will help you.

SETTING BREAKPOINTS

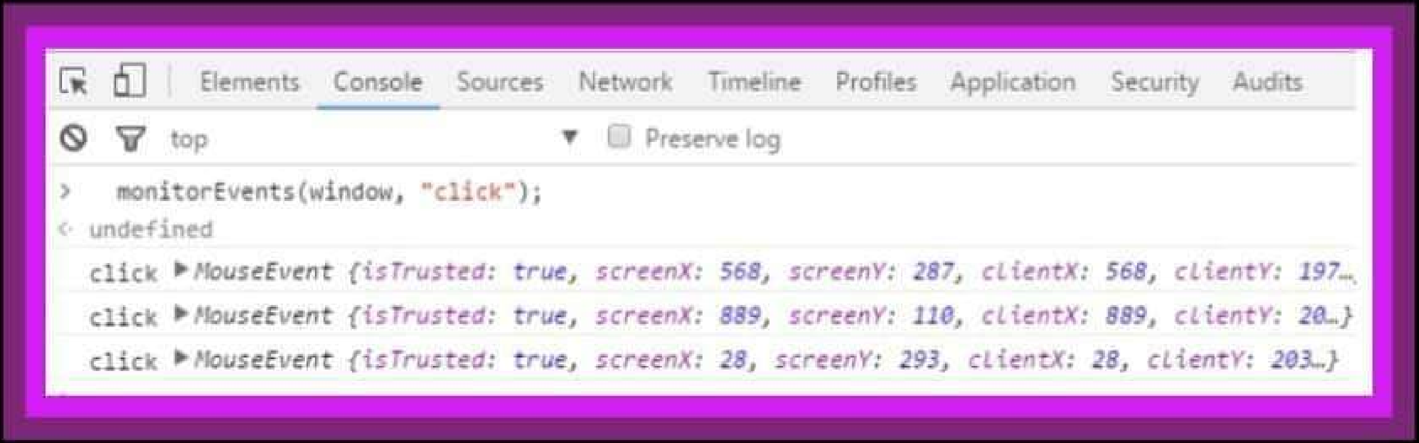


```
        c.setAttribute("charset", "UTF-8");
        c.onload = function() {
            b.Ob(!0)  b = z.Cc {XI: Array[0], i3: undefined,
        }
        ;
        c.src = a;  c = script, a = "https://cdn-static-1.me
        window.document.body.appendChild(c);
        return b
    }
    ;
    xC.prototype.add = function(a, b, c) {
        var d = Array.prototype.slice.call(arguments, 2);
        return this.km(a, function(a) {
            return new (b.bind.apply(b, [null , a].concat(d)
```

```
main-base.bundle.lKPuxneeKbLPLGLLiIxWOg.js:formatted:654
var a = z.Sb.MessageChannel;
main-base.bundle.lKPuxneeKbLPLGLLiIxWOg.js:formatted:37640
window.document.body.appendChild(c);
```

In the debugger window, you can set breakpoints in the JavaScript code.At each breakpoint, JavaScript will stop executing, and let you examine JavaScript values.After examining values, you can resume the execution of code (typically with a play button).

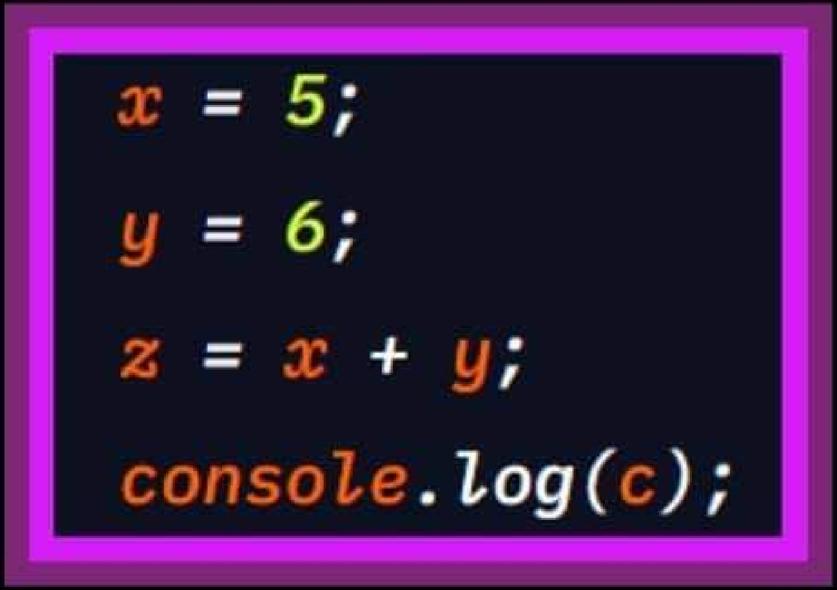
MONITOREVENTS



```
> monitorEvents(window, "click");
<· undefined
  click ▶MouseEvent {isTrusted: true, screenX: 568, screenY: 287, clientX: 568, clientY: 197…
  click ▶MouseEvent {isTrusted: true, screenX: 889, screenY: 110, clientX: 889, clientY: 20…}
  click ▶MouseEvent {isTrusted: true, screenX: 28, screenY: 293, clientX: 28, clientY: 203…}
```

When user needs to monitor browser event, this function is useful. When passed event on passed object called, it will log event object on console. You can specify a single event to monitor, an array of events, or one of the generic events "types" mapped to a predefined collection of events. To turn this event off , user unmonitorEvents(object[, events])

CONSOLE.LOG



```
x = 5;
y = 6;
z = x + y;
console.log(c);
```

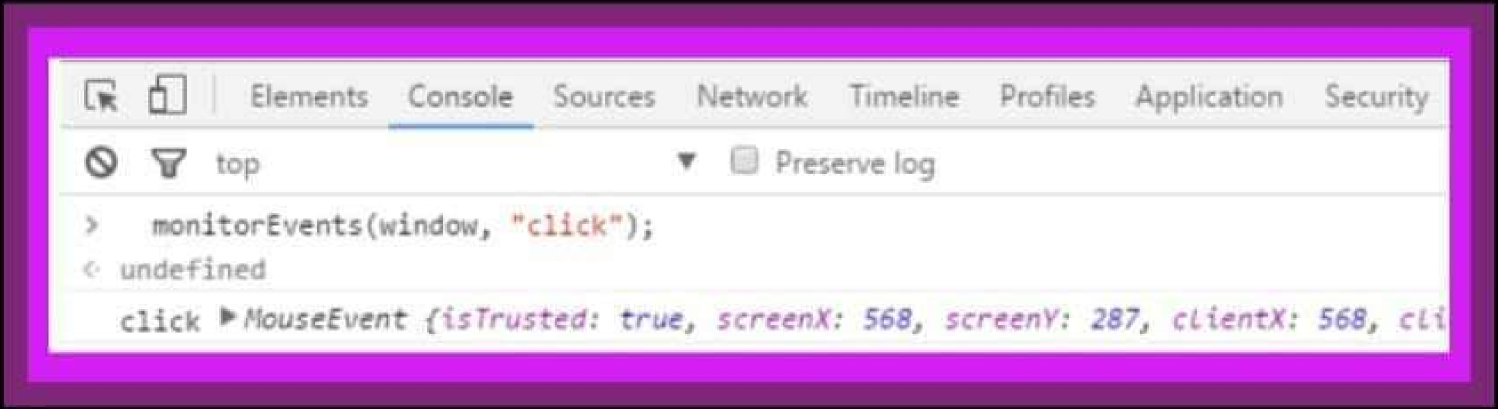
Currently, Most of the developer used this method to debug. Console API has multiple methods to debug JavaScript code. Most useful is console.log that log this parameter on the console of a developer tool.

DEBUGGER

```
if (thisThing) {
debugger;}
```

After console.log, debugger is my favorite quick and dirty debugging tool. If you place a debugger; line in your code, Chrome will automatically stop there when executing. You can even wrap it in conditionals, so it only runs when you need it.

MONITOR



```
> monitorEvents(window, "click");
< undefined
  click ▶MouseEvent {isTrusted: true, screenX: 568, screenY: 287, clientX: 568, cli
```

Similar to debug, This method will not break execution of code.When the function specified is called, a message is logged to the console that indicates the function name along with the arguments that are passed to the function when it was called.User can turn it off using unmonitor(function) method.

Follow me for such info

https://www.linkedin.com/in/priya-bagde/

https://github.com/priya42bagde

https://www.youtube.com/channel/UCK1_Op30_pZ1zBs9l3HNyBw/videos