

JavaScript

Какво си ти?

Ендонишков, асинхронен (non-blocking), паралелен език.

JavaScript работи чрез стек на извикванията (**call stack**), цикъл на събитията (**event loop**), опашка за обратно извикване (**callback queue**) и някои други **API**та.

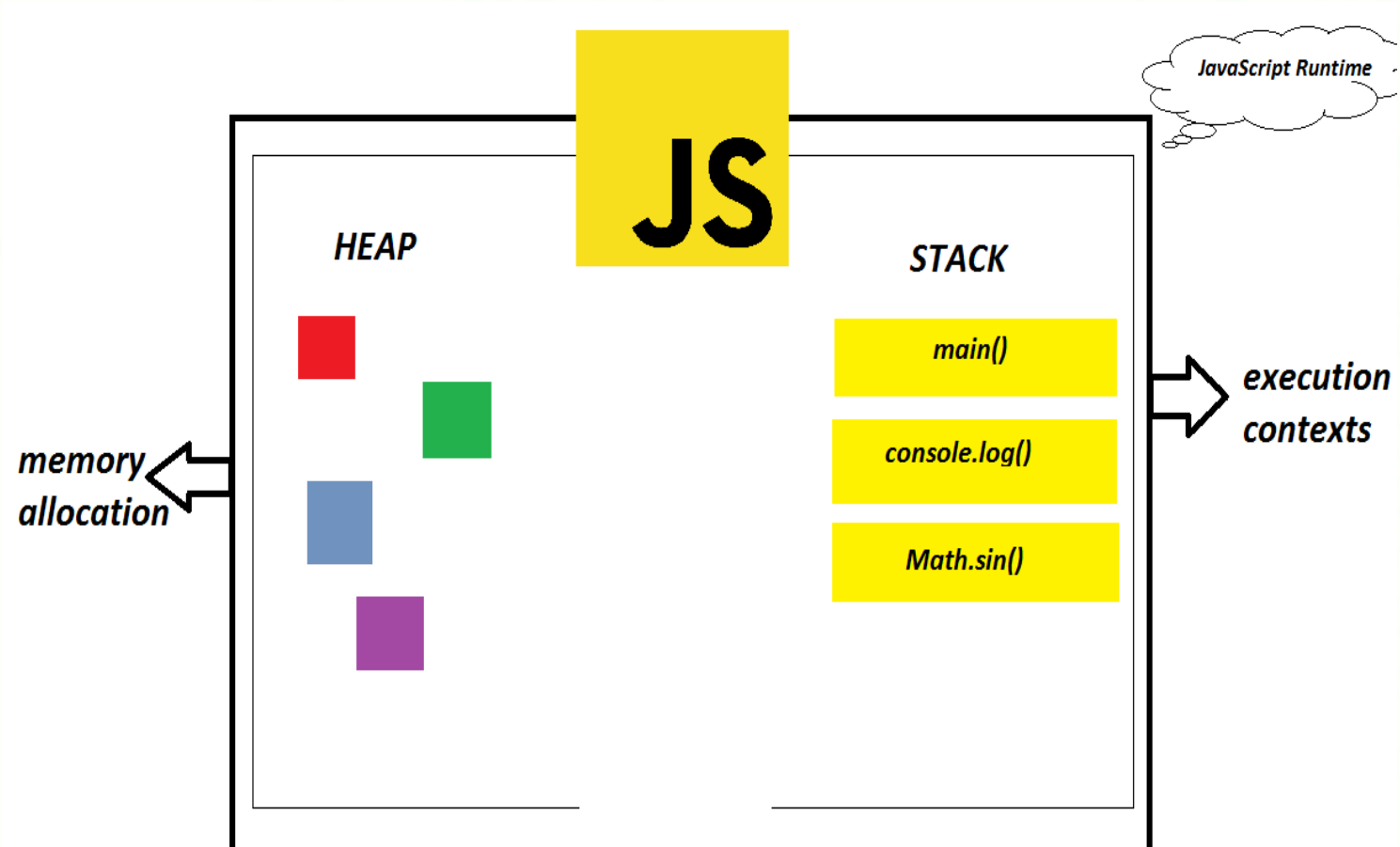


JavaScript базиран енджин разработен за уеб браузъра **Google Chrome**. (ползва се и в **MongoDB** и **Node.js** масово използвани на сървърите)

Какво всъщност е **JavaScript Engine**? Това е програма, която преобразува JavaScript код в машинен код, който микропроцесорите могат да разберат!

Той притежава call stack и heap!

JS



Call Stack

Стек на повиквания - записва къде точно се намираме в програмата.

```
function multiply(a, b) {  
    return a * b;  
}  
  
function square(n) {  
    return multiply(n, n);  
}  
  
function printSquare(n) {  
    var squared = square(n);  
    console.log(squared);  
}  
  
printSquare(4);
```

stack

Call Stack



```
function multiply(a, b) {  
    return a * b;  
}  
  
function square(n) {  
    return multiply(n, n);  
}  
  
function printSquare(n) {  
    var squared = square(n);  
    console.log(squared);  
}  
  
printSquare(4);
```

stack

main()

Call Stack



```
function multiply(a, b) {  
  return a * b;  
}  
  
function square(n) {  
  return multiply(n, n);  
}  
  
function printSquare(n) {  
  var squared = square(n);  
  console.log(squared);  
}  
  
printSquare(4);
```

stack

main()

Call Stack

```
function multiply(a, b) {  
    return a * b;  
}  
  
function square(n) {  
    return multiply(n, n);  
}  
  
function printSquare(n) {  
    var squared = square(n);  
    console.log(squared);  
}  
  
printSquare(4);
```

stack

main()

Call Stack

```
function multiply(a, b) {  
    return a * b;  
}  
  
function square(n) {  
    return multiply(n, n);  
}  
  
function printSquare(n) {  
    var squared = square(n);  
    console.log(squared);  
}  
  
→ printSquare(4);
```

Function call

stack

main()

Call Stack

```
function multiply(a, b) {  
    return a * b;  
}
```

```
function square(n) {  
    return multiply(n, n);  
}
```

→

```
function printSquare(n) {  
    var squared = square(n);  
    console.log(squared);  
}
```

```
printSquare(4);
```

stack

Ако извикаме функция –
добавяме я в стека,
Ако излизаме от функция,
премахваме от стека.

printSquare(4)

main()

Call Stack

```
function multiply(a, b) {  
    return a * b;  
}
```

→

```
function square(n) {  
    return multiply(n, n);  
}
```

```
function printSquare(n) {  
    var squared = square(n);  
    console.log(squared);  
}
```

```
printSquare(4);
```

stack

square(n)

printSquare(4)

main()

Call Stack



```
function multiply(a, b) {  
    return a * b;  
}
```

```
function square(n) {  
    return multiply(n, n);  
}
```

```
function printSquare(n) {  
    var squared = square(n);  
    console.log(squared);  
}
```

```
printSquare(4);
```

stack

multiply(n, n)

square(n)

printSquare(4)

main()

Call Stack

```
function multiply(a, b) {  
  return a * b;  
}
```

```
function square(n) {  
  return multiply(n, n);  
}
```

```
function printSquare(n) {  
  var squared = square(n);  
  console.log(squared);  
}
```

```
printSquare(4);
```

stack

Pop

square(n)


printSquare(4)

main()

Call Stack

```
function multiply(a, b) {  
    return a * b;  
}
```

```
function square(n) {  
    return multiply(n, n);  
}
```



```
function printSquare(n) {  
    var squared = square(n);  
    console.log(squared);  
}
```

```
printSquare(4);
```

stack

printSquare(4)

main()

Call Stack

```
function multiply(a, b) {  
    return a * b;  
}
```

```
function square(n) {  
    return multiply(n, n);  
}
```

```
function printSquare(n) {  
    var squared = square(n);  
    console.log(squared);  
}
```

```
printSquare(4);
```

stack

console.log(squared)


printSquare(4)

main()

Call Stack

```
function multiply(a, b) {  
    return a * b;  
}
```

```
function square(n) {  
    return multiply(n, n);  
}
```



```
function printSquare(n) {  
    var squared = square(n);  
    console.log(squared);  
}
```

```
printSquare(4);
```

stack

printSquare(4)

main()

Call Stack

```
function multiply(a, b) {  
    return a * b;  
}  
  
function square(n) {  
    return multiply(n, n);  
}  
  
function printSquare(n) {  
    var squared = square(n);  
    console.log(squared);  
}  
  
printSquare(4);
```

stack

Call Stack

```
function foo() {  
  throw new Error( 'Oops!' );  
}
```

```
function bar() {  
  foo();  
}
```

```
function baz() {  
  bar();  
}
```

```
baz();
```



The screenshot shows a web browser's developer console with the 'Elements' and 'Network' tabs selected. The console displays an uncaught error: 'Uncaught Error: Oops!'. The call stack is visible, showing the sequence of function calls that led to the error: 'foo' (oops.js:2), 'bar' (oops.js:7), 'baz' (oops.js:11), and '(anonymous function)' (oops.js:14). The console also shows a search icon, a filter icon, and a close button.

Call Stack Frame	File Name	Line Number
Uncaught Error: Oops!	oops.js	2
foo	oops.js	2
bar	oops.js	7
baz	oops.js	11
(anonymous function)	oops.js	14



Call Stack

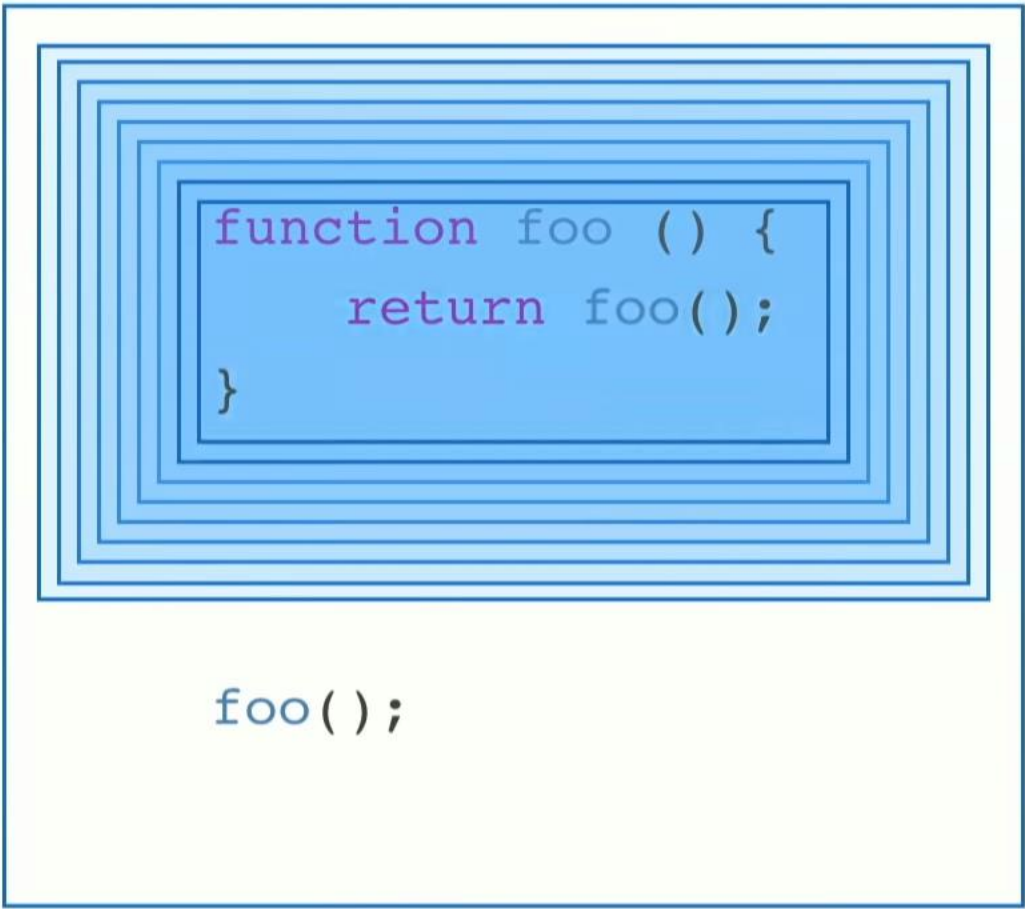
```
function foo () {  
    return foo();  
}
```

```
foo();
```

stack

A large, empty rectangular box with a thin black border, representing the call stack. The word 'stack' is written in the top right corner of the box.

Call Stack



Call Stack



X **RangeError: Maximum call stack size exceeded**

