

🚩 Current Skill Global Objects

__filename

Node.js global objects are global in nature and they are available in all modules. We do not need to include these objects in our application, but instead, we can use them directly. These objects are modules, functions, strings and objects themselves. We will explain more in the upcoming slides.

The `__filename` represents the filename of the code being executed. This is the resolved absolute path of the code file. For a main program, this is not necessarily the same filename used in the command line. The value inside a module is the path to that module's file.

Example

Create a .js file named **main.js** with the following code:

```
// Let's try to print the value of __filename  
console.log( __filename );
```

Now run the main.js to see the result:



```
$ node main.js
```

Based on the location of your program, it will print the main file name as follows:

```
/web/com/1427091028_21099/main.js
```

__dirname

The `__dirname` represents the name of the directory that the currently executing script resides in.



Example

Create a .js file named **main.js** with the following code:

```
// Let's try to print the value of __dirname  
console.log( __dirname );
```



Now run the main.js to see the result:

```
$ node main.js
```

Based on the location of your program, it will print current directory names as follows:

```
/web/com/1427091028_21099
```

setTimeout(cb, ms)

The **setTimeout(cb, ms)** global function is used to run **callback** (cb) after at least **ms milliseconds**.

The actual delay depends on external factors like OS timer granularity and system load. *A timer cannot span more than 24.8 days.*

Example

Create a .js file named main.js with the following code:

```
function printHello() {  
    console.log( "Hello, World!");  
}  
  
// Now call above function after 2 seconds  
setTimeout(printHello, 2000);
```

Now run the main.js to see the result:

```
$ node main.js
```

Verify that the output is printed as **after a little delay**:

```
Hello, World!
```



clearTimeout(t)

The **clearTimeout(t)** global function is used to stop a timer that was previously created with **setTimeout()**. Here **t** is the timer returned by the **setTimeout()** function.



Example

Create a .js file named main.js with the following code:

```
function printHello() {  
    console.log( "Hello, World!");  
}  
  
// Now call above function after 2 seconds  
var t = setTimeout(printHello, 2000);  
  
// Now clear the timer  
clearTimeout(t);
```

Now run the main.js to see the result:

```
$ node main.js
```

Verify the output where you will not find anything printed.

setInterval(cb, ms)

The **setInterval(cb, ms)** global function is used to run a callback **cb** repeatedly after at least a few ms milliseconds. The actual delay depends on external factors like OS timer granularity and system load. A timer cannot span more than 24.8 days.

- 📖 This function returns an opaque value that represents the timer which can be used to clear the timer using the function **clearInterval(t)**.

Example

Create a .js file named main.js with the following code:

```
function printHello() {  
    console.log( "Hello, World!");  
}  
  
// Now call above function after 2 seconds  
setInterval(printHello, 2000);
```

Now run the main.js to see the result:

```
$ node main.js
```

The above program will execute `printHello()` after every 2 second. Due to system limitation.



[< Previous](#)

[next >](#)

