Null vs Undefined vs Not Defined

Null vs **Undefined** vs **Not Defined**

Primitive Data type

Null vs **Undefined** vs **Not Defined**

Primitive Data type

Reference Error

Null vs **Undefined** vs **Not Defined**

Primitive Data type

Reference Error

The predefined data types provided by JavaScript language are known as primitive data types

Null vs **Undefined** vs **Not Defined**

Primitive Data type

Reference Error

The predefined data types provided by JavaScript language are known as primitive data types

Number

Null vs **Undefined** vs **Not Defined**

Primitive Data type

Reference Error

The predefined data types provided by JavaScript language are known as primitive data types

Number

String

Null vs **Undefined** vs **Not Defined**

Primitive Data type

Reference Error

The predefined data types provided by JavaScript language are known as primitive data types

Number String Boolean

Null vs Undefined vs Not Defined

Primitive Data type

Reference Error

The predefined data types provided by JavaScript language are known as primitive data types

Number String Boolean Bigint

Null vs **Undefined** vs **Not Defined**

Primitive Data type

Reference Error

The predefined data types provided by JavaScript language are known as primitive data types

Number String Boolean Bigint Symbol

Null vs **Undefined** vs **Not Defined**

Primitive Data type

Reference Error

The predefined data types provided by JavaScript language are known as primitive data types

Number String Boolean Bigint Symbol

Null

Null vs **Undefined** vs **Not Defined**

Primitive Data type

Reference Error

The predefined data types provided by JavaScript language are known as primitive data types

Number String Boolean Bigint Symbol

Null Undefined

Null vs Undefined vs Not Defined

Primitive Data type

Reference Error

```
> console.log(total)
Uncaught ReferenceError: total is not defined          VM211:1
    at <anonymous>:1:13
>
```

Undefined

undefined is a primitive value that is automatically assigned to variables that have been declared but have not been assigned a value

Undefined

undefined is a primitive value that is automatically assigned to variables that have been declared but have not been assigned a value

JS sample.js

Users > prakashshukla > Desktop > JS sample.js > ...

```
1
2   var result;
3   console.log(result);
4
5
6
```

Undefined

undefined is a primitive value that is automatically assigned to variables that have been declared but have not been assigned a value

JS sample.js

Users > prakashshukla > Desktop > JS sample.js > ...

```
1
2
3   var result = undefined;
4
5
6
```

Null

JavaScript primitive type `null` represents an intentional absence of a value.

Null

JavaScript primitive type `null` represents an intentional absence of a value.

JS sample.js

Users > prakashshukla > Desktop > JS sample.js > ...

```
1
2   var result = null;
```

Notdefined

In JavaScript, “not defined” refers to a reference error that occurs when attempting to access variables or identifiers that are not declared or do not exist within the current scope

Notdefined

In JavaScript, “not defined” refers to a reference error that occurs when attempting to access variables or identifiers that are not declared or do not exist within the current scope

JS sample.js

Users > prakashshukla > Desktop > JS sample.js

```
1
2   console.log(result);
3
4
5
```

Notdefined

In JavaScript, “not defined” refers to a reference error that occurs when attempting to access variables or identifiers that are not declared or do not exist within the current scope

JS sample.js

Users > prakashshukla > Deskto

```
1
2   console.log(result);
3
4
5
```

STDIN

Input for the program (Optional)

Output:

```
/index.js:1
console.log(result);
            ^

ReferenceError: result is not defined
    at Object.<anonymous> (/index.js:1:13)
    at Module._compile (node:internal/modules/cjs/loac
    at Object.Module._extensions..js (node:internal/mc
    at Module.load (node:internal/modules/cjs/loader:9
    at Function.Module._load (node:internal/modules/cj
    at Function.executeUserEntryPoint [as runMain] (nc
    at node:internal/main/run_main_module:17:47
```

PPT Notes in Description

Hoisting

Hoisting is built-in behavior of the language through which declarations of functions, variables, and classes are moved to the top of their scope – all before code execution.

Hoisting

Allows us to use functions, variables, and classes before they are declared.

Hoisting

(Hoist a flag or raise a flag)

Hoisting

Functions

Variables with var , let and const

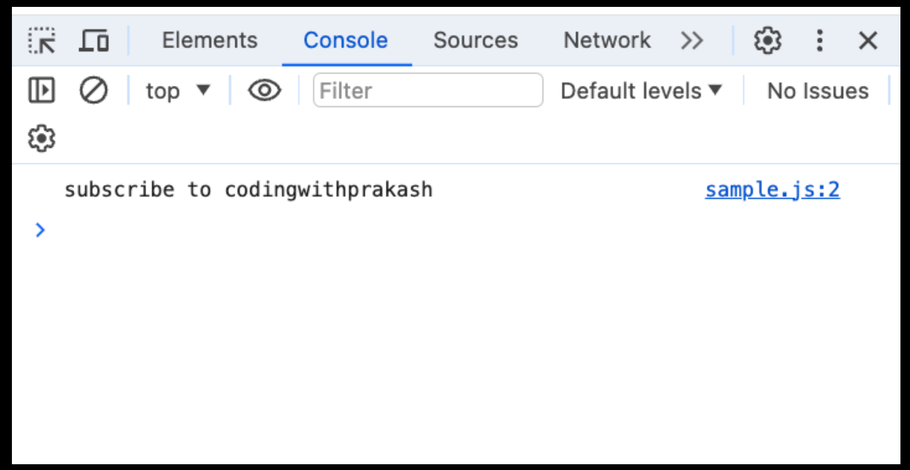
Classes

Function Hoisting

```
function subscribe() {
  console.log("subscribe to codingwithprakash");
}

subscribe();
```

Function Hoisting

```
function subscribe() {
  console.log("subscribe to codingwithprakash");
}

subscribe();
```



Function Hoisting

```
subscribe();

function subscribe() {
  console.log("subscribe to codingwithprakash");
}
```

Function Hoisting

```
subscribe();

function subscribe() {
  console.log("subscribe to codingwithprakash");
}
```

function declarations are put into memory at compile time

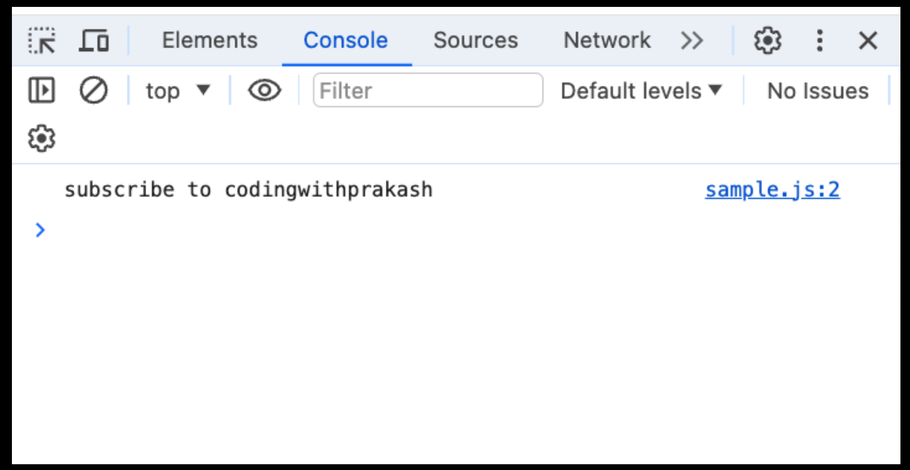
It hoists a function's declaration and its value to the top of their scope.

Function Hoisting

function declarations are put into memory at compile time

```
subscribe();

function subscribe() {
  console.log("subscribe to codingwithprakash");
}
```

It hoists a function's declaration and its value to the top of their scope.

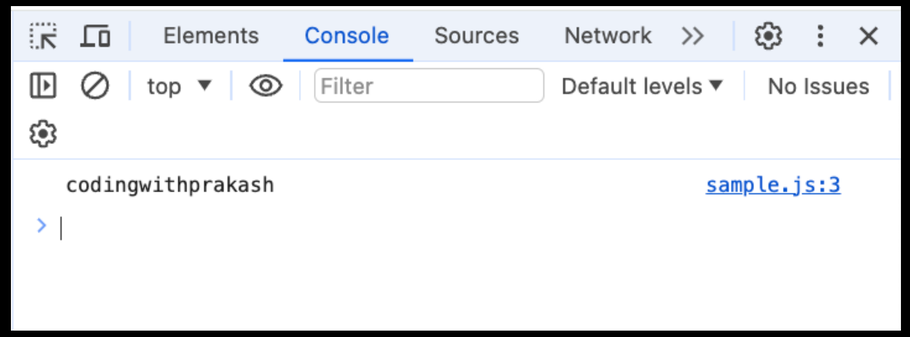


Variable Hoisting (var)

```
var channelName = "codingwithprakash";

console.log(channelName);
```

Variable Hoisting (var)

```
var channelName = "codingwithprakash";

console.log(channelName);
```

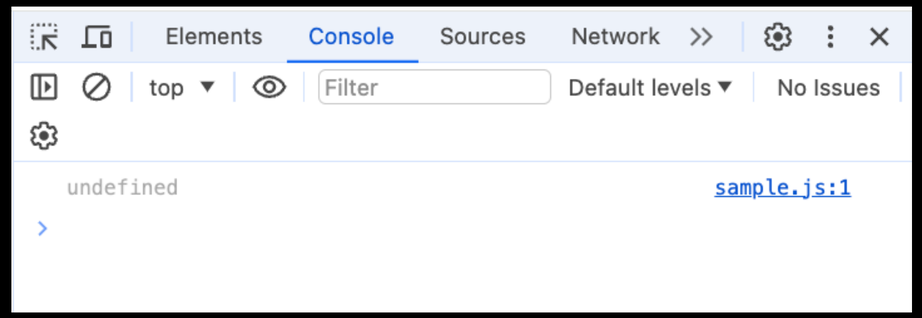


Variable Hoisting (var)

```
console.log(channelName);

var channelName = "codingwithprakash";
```

Variable Hoisting (var)

```
console.log(channelName);

var channelName = "codingwithprakash";
```

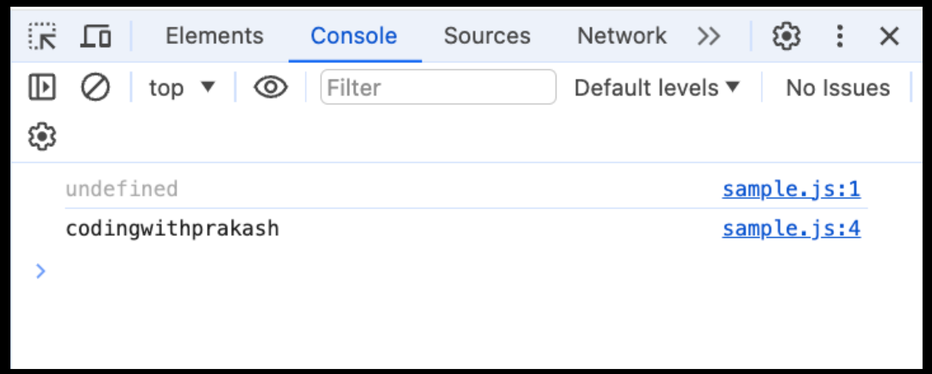


Variable Hoisting (var)

```
console.log(channelName);

var channelName = "codingwithprakash";

console.log(channelName);
```

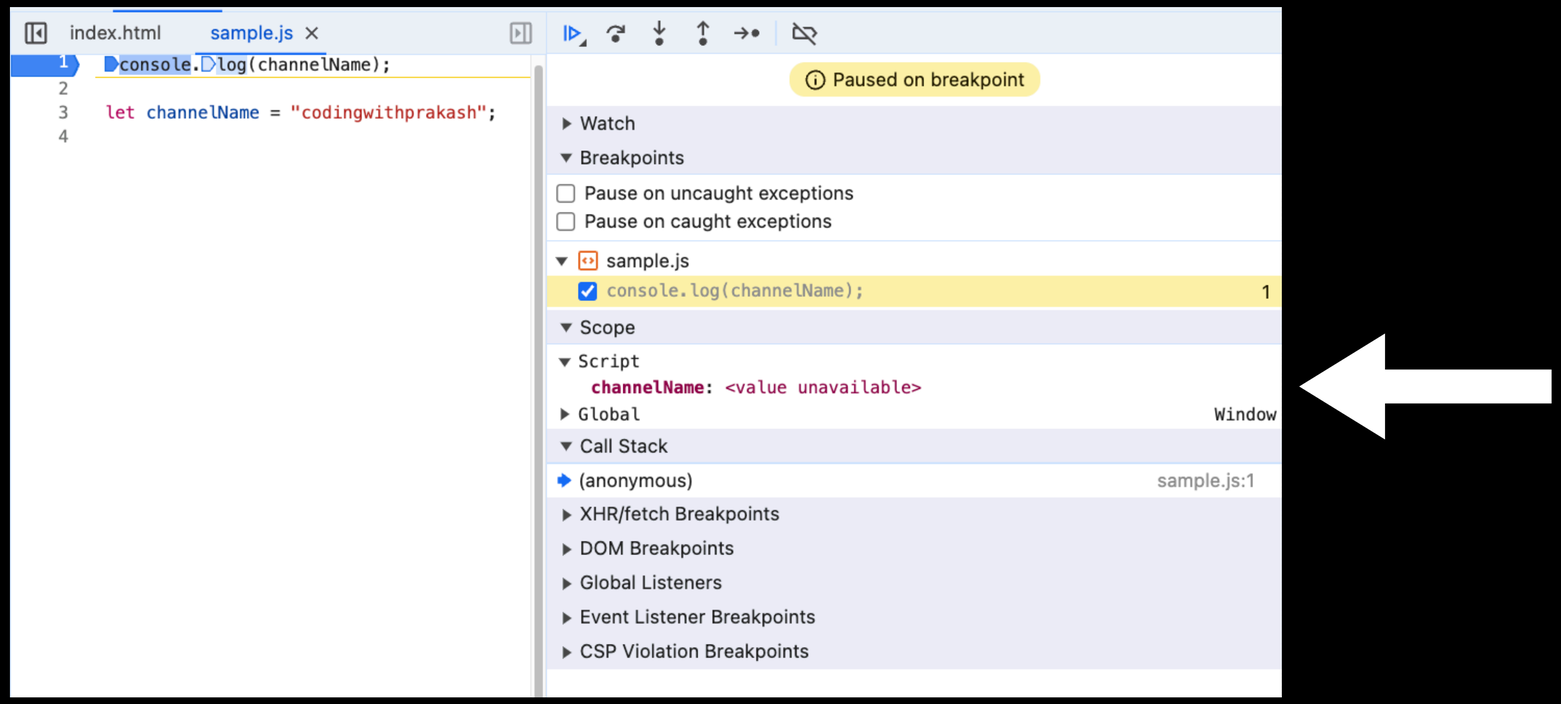
Variable Hoisting (var)

```
console.log(channelName);

var channelName = "codingwithprakash";

console.log(channelName);
```

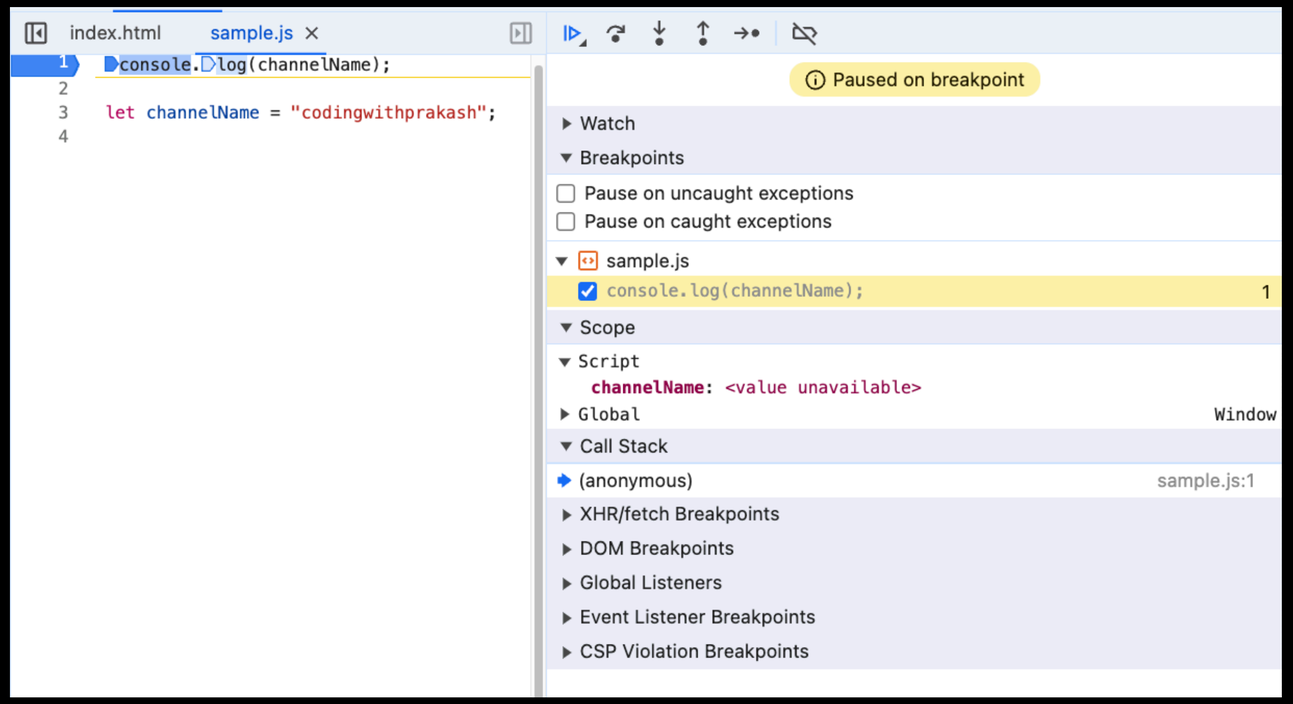


Variable Hoisting (let and const)

Variable Hoisting (let and const)

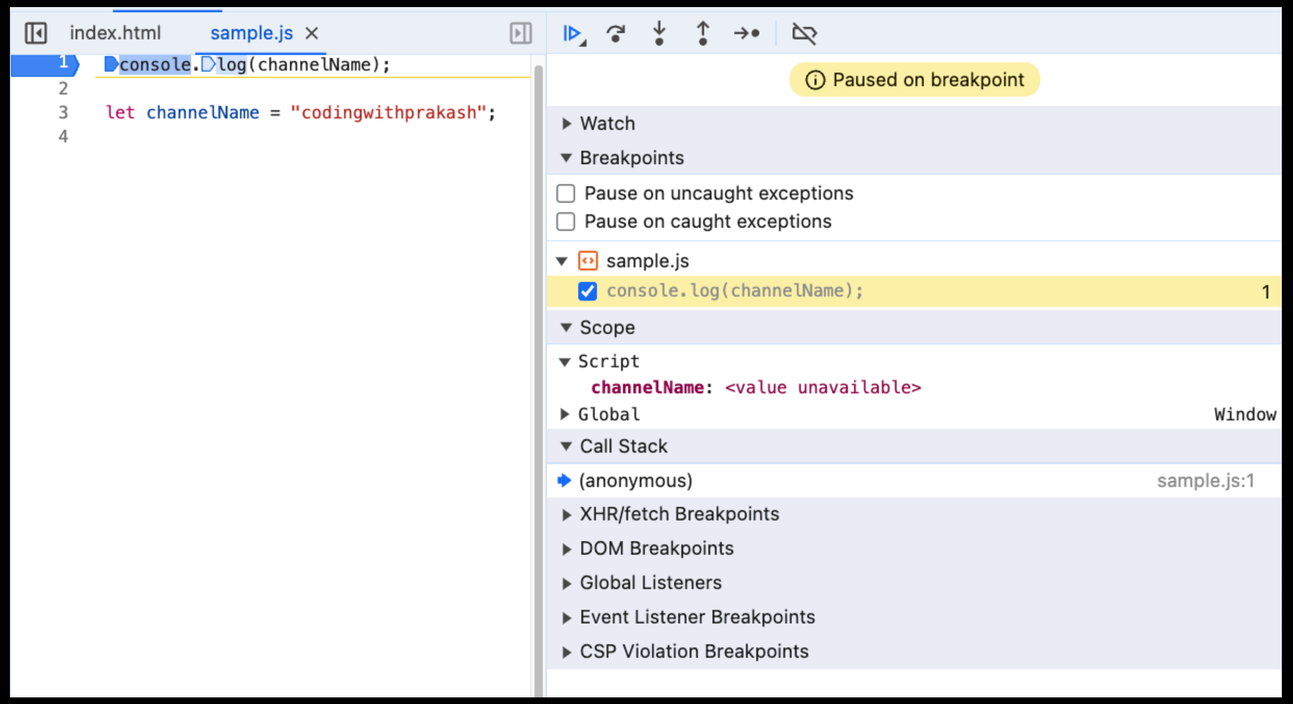


Variable Hoisting (let and const)



Temporal Dead Zone

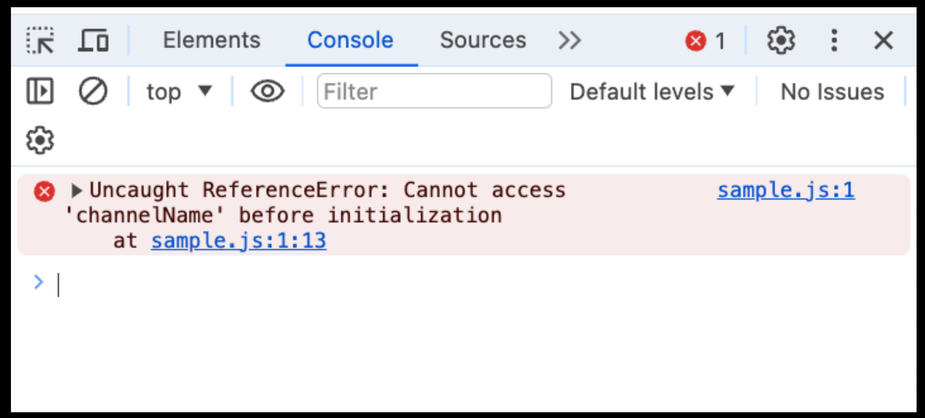
Variable Hoisting (let and const)



Temporal Dead Zone

During this phase, the variable exists but is inaccessible. Accessing it before the declaration results in a ReferenceError.

Variable Hoisting (let and const)



Variable Hoisting (let and const)

Code reliability

Predictability

Class Hoisting

Users > prakashshukla > Desktop > JS sample.js > ...

```
1
2   class Vehicle {}
3
4
5
6
```

Class Hoisting

Users > prakashshukla > Desktop > JS sample.js > ...

```
1
2   class Vehicle {}
3
4
5
6
```

Users > prakashshukla > Desktop > JS sample.js > ...

```
1
2
3   let instance = new Vehicle();
4
5   class Vehicle {}
6
7
```

Class Hoisting

Users > prakashshukla > Desktop > JS sample.js > ...

```
1
2   class Vehicle {}
3
4
5
6
```

Elements | Console | Sources | Network | Performance | Memory

top ▼ | Filter

```
Uncaught ReferenceError: Cannot access 'Vehicle' before initialization
    at sample.js:1:16
>
```

Users > prakashshukla > Desktop > JS sample.js > ...

```
1
2
3   let instance = new Vehicle();
4
5   class Vehicle {}
6
7
```

Hoisting

Declaration and its value	Only Declaration	Hoisted but in TDZ
Functions	Variable with Var	Variable with let and const Classes