A let or const variable is said to be in a "temporal dead zone" (TDZ) from the start of the block until code execution reaches the line where the variable is declared and initialized.

While inside the TDZ, the variable has not been initialized with a value, and any attempt to access it will result in a ReferenceError . The variable is initialized with a value when execution reaches the line of code where it was declared. If no initial value was specified with the variable declaration, it will be initialized with a value of undefined .

This differs from var variables, which will return a value of undefined if they are accessed before they are declared. The code below demonstrates the different result when let and var are accessed in code before the line in which they are declared.

```
{ console . log ( bar ) ; console . log ( foo ) ; var bar = 1 ; let foo = 2 ; }
```

The term "temporal" is used because the zone depends on the order of execution (time) rather than the order in which the code is written (position). For example, the code below works because, even though the function that uses the let variable appears before the variable is declared, the function is called outside the TDZ.

```
{ const func = ( ) => console . log ( letVar ) ; let letVar = 3 ; func ( ) ; }
```

The TDZ and typeof

Using the typeof operator for a let variable in its TDZ will throw a ReferenceError :

```
console . log ( typeof i ) ; let i = 10 ;
```

This differs from using typeof for undeclared variables, and variables that hold a value of undefined :

```
console . log ( typeof undeclaredVariable ) ;
```

TDZ combined with lexical scoping

The following code results in a ReferenceError at the line shown:

```
function test ( ) { var foo = 33 ; if ( foo ) { let foo = foo + 55 ; } } test ( ) ;
```

The if block is evaluated because the outer var foo has a value. However due to lexical scoping this value is not available inside the block: the identifier foo inside the if block is the let foo . The expression (foo + 55) throws a ReferenceError because initialization of let foo has not completed â€” it is still in the temporal dead zone.

P

This phenomenon can be confusing in a situation like the following. The instruction let n of n.a is already inside the private scope of the for loop's block. So, the identifier n.a is resolved to the property ' a ' of the ' n ' object located in the first part of the instruction itself (let n).

This is still in the temporal dead zone as its declaration statement has not been reached and terminated.

Reference

[The Public Shaping of Medical Research: Patient Associations, Health Movements and Biomedicine (Routledge Studies in the Sociology of Health and Illness)]

[Transformative Research and Evaluation]