### Assignment 2

### [Temporal dead zone (TDZ)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/let#temporal_dead_zone_tdz)

### let variables cannot be read/written until they have been fully initialized, which happens when they are declared (if no initial value is specified on declaration, the variable is initialized with a value of undefined). Accessing the variable before the initialization results in a [ReferenceError](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/ReferenceError).

### **Note:** This differs from [var](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/var#var_hoisting) variables, which will return a value of undefined if they are accessed before they are declared.

### The variable is said to be in a "temporal dead zone" (TDZ) from the start of the block until the initialization has completed.

### { // TDZ starts at beginning of scope

### console.log(bar); // undefined

### console.log(foo); // ReferenceError

### var bar = 1;

### let foo = 2; // End of TDZ (for foo)

### }

### 

### Copy to Clipboard

### 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.

### {

### // TDZ starts at beginning of scope

### const func = () => console.log(letVar); // OK

### 

### // Within the TDZ letVar access throws `ReferenceError`

### 

### let letVar = 3; // End of TDZ (for letVar)

### func(); // Called outside TDZ!

### }

### 

### Copy to Clipboard

#### The TDZ and typeof

### Using the typeof operator for a let variable in its TDZ will throw a [ReferenceError](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/ReferenceError):

### // results in a 'ReferenceError'

### console.log(typeof i);

### let i = 10;

### 

### Copy to Clipboard

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

### // prints out 'undefined'

### console.log(typeof undeclaredVariable);

### 

### Copy to Clipboard

#### 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); // ReferenceError

### }

### }

### test();

### 

### Copy to Clipboard

### 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 — it is still in the temporal dead zone.

### 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.

### function go(n) {

### // n here is defined!

### console.log(n); // Object {a: [1,2,3]}

### 

### for (let n of n.a) { // ReferenceError

### console.log(n);

### }

### }

### 

### go({a: [1, 2, 3]});

### 

### Copy to Clipboard

### [Other situations](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/let#other_situations)

### When used inside a block, **let** limits the variable's scope to that block. Note the difference between **var**, whose scope is inside the function where it is declared.

### var a = 1;

### var b = 2;

### 

### if (a === 1) {

### var a = 11; // the scope is global

### let b = 22; // the scope is inside the if-block

### 

### console.log(a); // 11

### console.log(b); // 22

### }

### 

### console.log(a); // 11

### console.log(b); // 2

### 

### Copy to Clipboard

### However, this combination of **var** and **let** declaration below is a [SyntaxError](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/SyntaxError) due to **var** being hoisted to the top of the block. This results in an implicit re-declaration of the variable.

### let x = 1;

### 

### {

### var x = 2; // SyntaxError for re-declaration

### }

### 

### Copy to Clipboard

## [**Specifications**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/let#specifications)

| **Specification** |
| --- |
| [ECMAScript Language Specification (ECMAScript)](https://tc39.es/ecma262/multipage/ecmascript-language-statements-and-declarations.html#sec-let-and-const-declarations)[# sec-let-and-const-declarations](https://tc39.es/ecma262/multipage/ecmascript-language-statements-and-declarations.html#sec-let-and-const-declarations) |

## [**Browser compatibility**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/let#browser_compatibility)

### [Report problems with this compatibility data on GitHub](https://github.com/mdn/browser-compat-data/issues/new?body=%3C%21--+Tips%3A+where+applicable%2C+specify+browser+name%2C+browser+version%2C+and+mobile+operating+system+version+--%3E%0A%0A%23%23%23%23+What+information+was+incorrect%2C+unhelpful%2C+or+incomplete%3F%0A%0A%23%23%23%23+What+did+you+expect+to+see%3F%0A%0A%23%23%23%23+Did+you+test+this%3F+If+so%2C+how%3F%0A%0A%0A%3C%21--+Do+not+make+changes+below+this+line+--%3E%0A%3Cdetails%3E%0A%3Csummary%3EMDN+page+report+details%3C%2Fsummary%3E%0A%0A*+Query%3A+%60javascript.statements.let%60%0A*+MDN+URL%3A+https%3A%2F%2Fdeveloper.mozilla.org%2Fen-US%2Fdocs%2FWeb%2FJavaScript%2FReference%2FStatements%2Flet%0A*+Report+started%3A+2021-08-06T18%3A54%3A24.381Z%0A%0A%3C%2Fdetails%3E&title=javascript.statements.let+-+%3CPUT+TITLE+HERE%3E)

|  | **desktop** | | | | | | **mobile** | | | | | | **server** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Chrome | Edge | Firefox | Internet Explorer | Opera | Safari | WebView Android | Chrome Android | Firefox for Android | Opera Android | Safari on iOS | Samsung Internet | Node.js |
| let | Full support49Open | Full support14Open | Full support44footnoteOpen | Partial support11footnoteOpen | Full support17 | Full support10 | Full support49Open | Full support49Open | Full support44footnoteOpen | Full support18 | Full support10 | Full support5.0Open | Full support6.0.0 |

### Legend

### **Full support**

### Full support

### **Partial support**

### Partial support

### **No support**

### No support

### See implementation notes.

### User must explicitly enable this feature.

### To try and access a variable before it is declared is the wrong way round, and shouldn't be possible.

### It also gives more expected and rational semantics for const (because const is hoisted, what happens if a programmer tries to use it before it is declared at runtime? What variable should it hold at the point when it gets hoisted?), and was the best approach decided by the ECMAScript spec team.

## **How to avoid the issues the TDZ causes**

### Relatively simply, always make sure you define your lets and consts at the top of your scope.

### 

### [**Temporal dead zone (TDZ)**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/let#temporal_dead_zone_tdz)

let variables cannot be read/written until they have been fully initialized, which happens when they are declared (if no initial value is specified on declaration, the variable is initialized with a value of undefined). Accessing the variable before the initialization results in a [ReferenceError](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/ReferenceError).

**Note:** This differs from [var](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/var#var_hoisting) variables, which will return a value of undefined if they are accessed before they are declared.

The variable is said to be in a "temporal dead zone" (TDZ) from the start of the block until the initialization has completed.

{ // TDZ starts at beginning of scope

console.log(bar); // undefined

console.log(foo); // ReferenceError

var bar = 1;

let foo = 2; // End of TDZ (for foo)

}

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.

{

// TDZ starts at beginning of scope

const func = () => console.log(letVar); // OK

// Within the TDZ letVar access throws `ReferenceError`

let letVar = 3; // End of TDZ (for letVar)

func(); // Called outside TDZ!

}

#### 

#### 

#### 

#### **The TDZ and typeof**

Using the typeof operator for a let variable in its TDZ will throw a [ReferenceError](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/ReferenceError):

// results in 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:

// prints out '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); // ReferenceError

}

}

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 — it is still in the temporal dead zone.

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.

function go(n) {

// n here is defined!

console.log(n); // Object {a: [1,2,3]}

for (let n of n.a) { // ReferenceError

console.log(n);

}

}

go({a: [1, 2, 3]});

### [Other situations](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/let#other_situations)

When used inside a block, **let** limits the variable's scope to that block. Note the difference between **var**, whose scope is inside the function where it is declared.

var a = 1;

var b = 2;

if (a === 1) {

var a = 11; // the scope is global

let b = 22; // the scope is inside the if-block

console.log(a); // 11

console.log(b); // 22

}

console.log(a); // 11

console.log(b); // 2

However, this combination of **var** and **let** declaration below is a [SyntaxError](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/SyntaxError) due to **var** being hoisted to the top of the block. This results in an implicit re-declaration of the variable.

let x = 1;

{

var x = 2; // SyntaxError for re-declaration

}

## [**Specifications**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/let#specifications)

| **Specification** |
| --- |
| [ECMAScript Language Specification (ECMAScript)](https://tc39.es/ecma262/multipage/ecmascript-language-statements-and-declarations.html#sec-let-and-const-declarations)  [# sec-let-and-const-declarations](https://tc39.es/ecma262/multipage/ecmascript-language-statements-and-declarations.html#sec-let-and-const-declarations) |

## [**Browser compatibility**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/let#browser_compatibility)

[Report problems with this compatibility data on GitHub](https://github.com/mdn/browser-compat-data/issues/new?body=%3C%21--+Tips%3A+where+applicable%2C+specify+browser+name%2C+browser+version%2C+and+mobile+operating+system+version+--%3E%0A%0A%23%23%23%23+What+information+was+incorrect%2C+unhelpful%2C+or+incomplete%3F%0A%0A%23%23%23%23+What+did+you+expect+to+see%3F%0A%0A%23%23%23%23+Did+you+test+this%3F+If+so%2C+how%3F%0A%0A%0A%3C%21--+Do+not+make+changes+below+this+line+--%3E%0A%3Cdetails%3E%0A%3Csummary%3EMDN+page+report+details%3C%2Fsummary%3E%0A%0A*+Query%3A+%60javascript.statements.let%60%0A*+MDN+URL%3A+https%3A%2F%2Fdeveloper.mozilla.org%2Fen-US%2Fdocs%2FWeb%2FJavaScript%2FReference%2FStatements%2Flet%0A*+Report+started%3A+2021-08-06T18%3A54%3A24.381Z%0A%0A%3C%2Fdetails%3E&title=javascript.statements.let+-+%3CPUT+TITLE+HERE%3E)

|  | **desktop** | | | | | | **mobile** | | | | | | **server** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Chrome | Edge | Firefox | Internet Explorer | Opera | Safari | WebView Android | Chrome Android | Firefox for Android | Opera Android | Safari on iOS | Samsung Internet | Node.js |
| let | Full support  49  Open | Full support  14  Open | Full support  44  footnote  Open | Partial support  11  footnote  Open | Full support  17 | Full support  10 | Full support  49  Open | Full support  49  Open | Full support  44  footnote  Open | Full support  18 | Full support  10 | Full support  5.0  Open | Full support  6.0.0 |

### Legend

**Full support**

Full support

**Partial support**

Partial support

**No support**

No support

See implementation notes.

User must explicitly enable this feature.