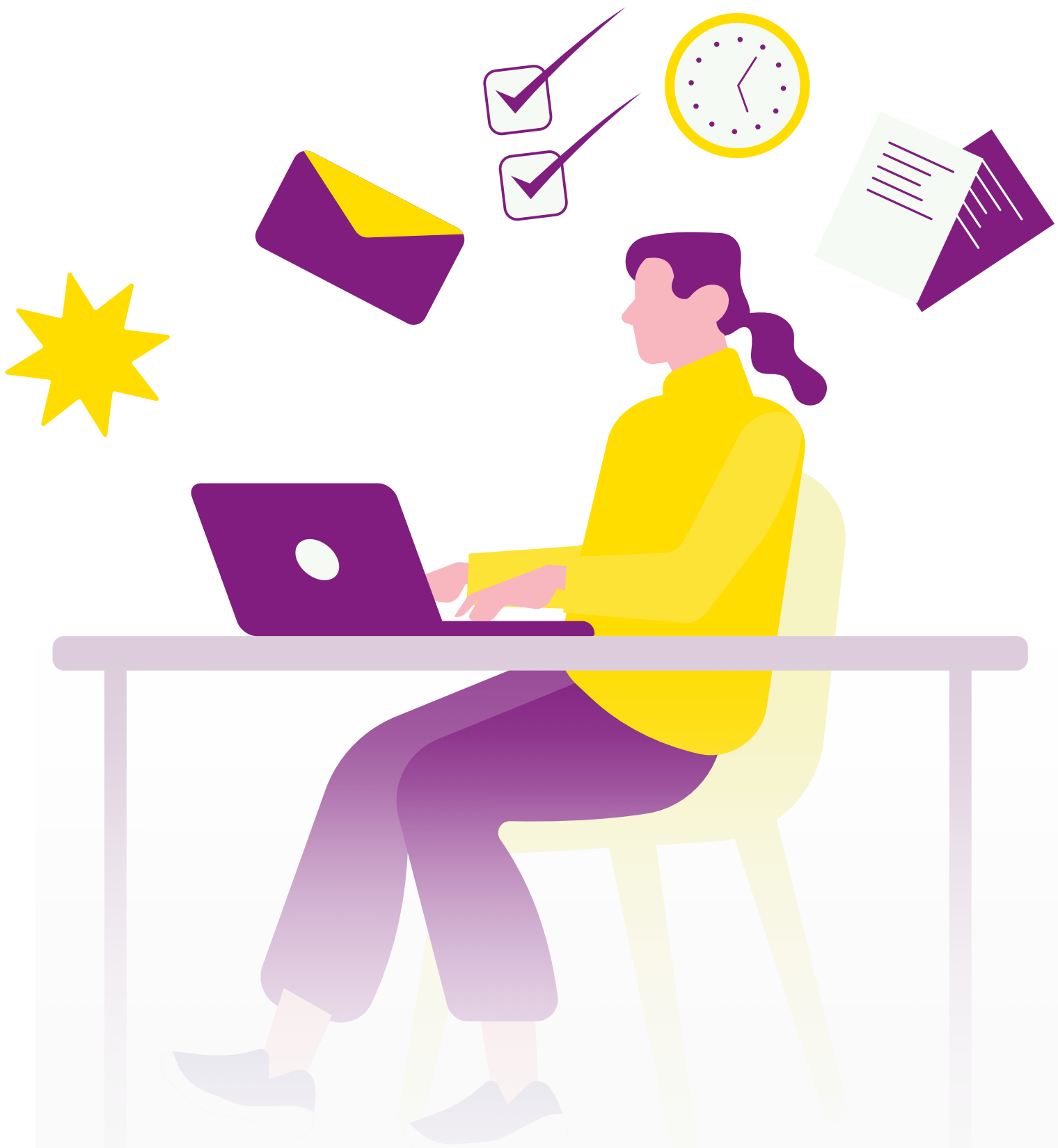
Salesforce LWC & JavaScript Enthusiasts

# VAR, LET AND CONST



Salesforce LWC & JavaScript Enthusiasts

01

# VAR

- Scope: Function-scoped, meaning if it's declared inside a function, it will be available throughout the entire function.
- Re-declaration: You can re-declare a var variable within the same scope without any errors.
- Hoisting: var declarations are hoisted to the top of their scope, meaning they can be accessed even before they are declared (but the value will be undefined until assignment)

```
1  var x = 10;
2  if (true) {
3      var x = 20;
4      console.log(x);
5  }
6  console.log(x);
```

```
Console ×

20
20
```

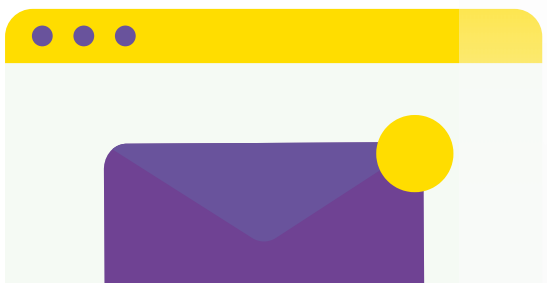
Salesforce LWC & JavaScript Enthusiasts

02

# LET

- Scope: Block-scoped, meaning it is only available within the block (e.g., {}) in which it was declared.
- Re-declaration: You cannot re-declare a let variable in the same scope, but you can reassign its value.
- Hoisting: Like var, let is hoisted, but it's not initialized. Accessing it before declaration results in a ReferenceError

```
let x = 10;
if (true) {
    let x = 20;
    console.log(x);
}
console.log(x);
```



```
Console ×

20

10
```

Salesforce LWC & JavaScript Enthusiasts

# CONST

03

- Scope: Block-scoped, similar to let.
- Re-declaration & Re-assignment: You cannot re-declare or reassign a const variable. Once a value is assigned, it can't be changed. However, if the variable holds an object or array, the properties of the object or elements of the array can be changed (but the reference cannot).
- Hoisting: Like let, const is hoisted but not initialized, and accessing it before declaration results in a ReferenceError

```
const car = {
    brand: 'Tesla',
    model: 'Model S',
    year: 2024
};
console.log(car.brand);
console.log(car.model);
console.log(car.year);
```

```
Console ×

Tesla

Model S

2024
```