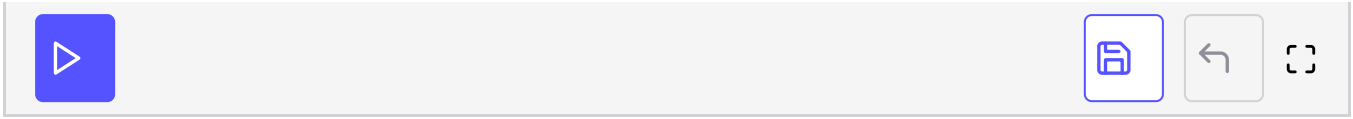
Default Function Parameters

We'll see how the new behavior of default function parameters really works. We'll explore its capabilities and see how it helps us reduce the amount of code we need to write.

Default Parameters

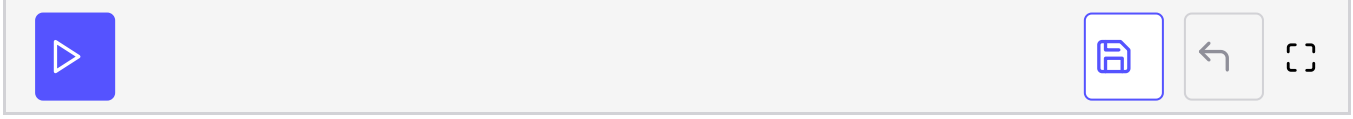
The old way to check for function parameters was to do so manually in the function body. If we had to make sure a parameter wasn't `undefined`, this is something we'd do.

```
function fn(param) {
    if (param === undefined) {
        param = "Default value";
    }

    console.log(param);
}

fn('String passed in'); // -> String passed in
fn(); // -> Default value
```



This is now a little easier.

```
function fn(param = "Default value") {
    console.log(param);
}

fn('String passed in'); // -> String passed in
fn(); // -> Default value
```

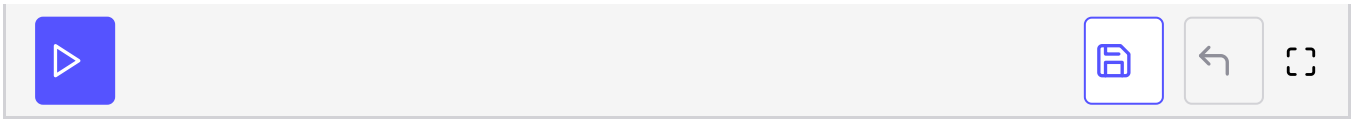


This syntax will check `param` when the function is called. If it was undefined meaning a user either passed in `undefined` or nothing at all, it will get the default value provided after the `=`.

Expressions as Parameters

In addition to static values, expressions can be used as default parameters.

```
function fn(param = 10 * 10) {
    console.log(param);
    return param;
}

function fn2(param = fn(50)) {
    console.log(param);
}

fn('String passed in'); // -> String passed in
fn(); // -> 100

fn2(); // -> 50
       // -> 50
```



Variable availability

Parameters are available in order to other parameters. A parameter to a function can access parameters that came before it.

```
function add(param1 = 10, param2 = param1) {
    console.log(param1 + param2);
}

add(2, 5); // -> 7
add(2); // -> 4
add(); // -> 20
```