Operations on All Elements

In this lesson, we will see operations on all elements of an array. This feature is for both fixed-length arrays and slices.

WE'LL COVER THE FOLLOWING

Operations on all elements of an array

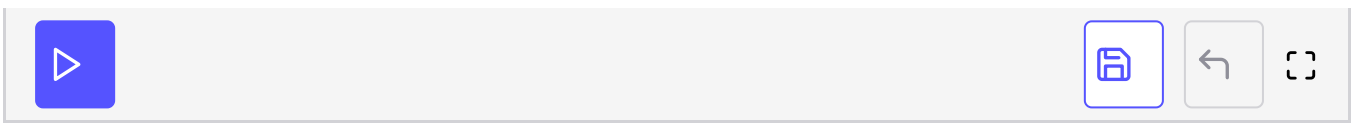
The `[]` characters written after the name of an array refers to all elements of the array. This feature simplifies the program when certain operations need to be applied to all of the elements of the array.

```
import std.stdio;

void main() {
    double[3] a = [ 10, 20, 30 ];
    double[3] b = [  2,  3,  4 ];

    double[3] result = a[] + b[];

    writeln(result);
}
```



Operation applied to all elements of an array

The addition operation in this program is applied to the corresponding elements of both of the arrays in order: the first two elements are added, then the second two elements are added and so forth. A natural requirement is that the lengths of the two arrays must be equal.

Operators that can be used on entire arrays include arithmetic operators (`+`,

`-`, `*`, `/`, `%`), binary operators (`^`, `&`, and `|`), as well as unary operators (`-` and `~`).

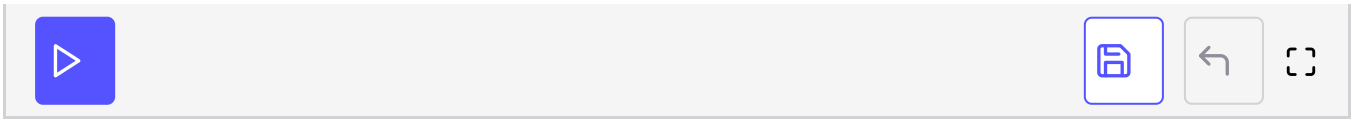
The shorthand versions of the above operators with assignments can also be used: `=`, `+=`, `-=`, `*=`, `/=`, `%=`, `^^=`, `^=`, `&=` and `|=`.
This feature not only works with array operands on both sides but can also be used with an array and a compatible expression. For example, the following operation divides all elements of an array by four:

```
import std.stdio;

void main(){
    double[3] a = [ 10, 20, 30 ];

    a[] /= 4;

    writeln(a);
}
```



Using /= operator on all elements of an array

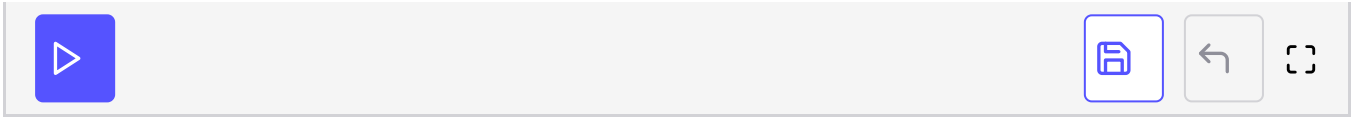
To assign a specific value to all elements:

```
import std.stdio;

void main(){
    double[3] a = [ 10, 20, 30 ];

    a[] = 42;

    writeln(a);
}
```



Using /= operator on all elements of an array

Operations on all elements of a slice

This feature requires great care when used with slices. Although there is no apparent difference in element values, the following two expressions have very different meanings:

```
slice2 = slice1;
```

```
// slice2 starts providing access to the same elements that slice1 provide
s access to

slice3[] = slice1;
// the values of the elements of slice3 change
```

The assignment of `slice2` makes it share the same elements as `slice1`. On the other hand, since `slice3[]` means all elements of `slice3`, the values of its elements become the same as the values of the elements of `slice1`. The effect of the presence or absence of the `[]` characters cannot be ignored.
We can see an example of this difference in the following program:

```
import std.stdio;

void main() {
    double[] slice1 = [ 1, 1, 1 ];
    double[] slice2 = [ 2, 2, 2 ];
    double[] slice3 = [ 3, 3, 3 ];

    slice2 = slice1;      // ← slice2 starts providing access
                          //   to the same elements that
                          //   slice1 provides access to

    slice3[] = slice1;    // ← the values of the elements of
                          //   slice3 change

    writeln("slice1 before: ", slice1);
    writeln("slice2 before: ", slice2);
    writeln("slice3 before: ", slice3);

    slice2[0] = 42;       // ← the value of an element that
                          //   it shares with slice1 changes

    slice3[0] = 43;       // ← the value of an element that
                          //   only it provides access to
                          //   changes

    writeln("slice1 after : ", slice1);
    writeln("slice2 after : ", slice2);
    writeln("slice3 after : ", slice3);
}
```

Effect of presence of [ ] characters

As it can be seen in the output of the above code, the modification through `slice2` affects `slice1`, too.

In the next lesson, we will learn about multi-dimensional arrays.