foreach Loop Properties

This lesson explains the different properties of foreach loop.

WE'LL COVER THE FOLLOWING

Counter for containers

The automatic counter in `foreach` is provided only when iterating over arrays. There are two options for other containers:

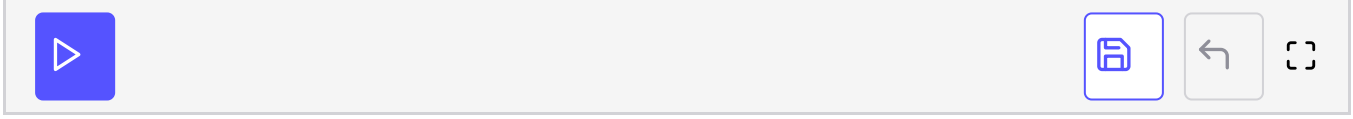
- Taking advantage of `std.range.enumerate`
- Defining and incrementing a counter variable explicitly:

```
size_t i = 0;

foreach (element; container) {
    // ...
    ++i;
}
```

A counter variable is needed when counting a specific condition as well. For example, the following code only counts the values that are divisible by 10 and displays the count with each occurrence of such an element.

```
import std.stdio;

void main() {
    auto numbers = [ 1, 0, 15, 10, 3, 5, 20, 30 ];
```

```
    size_t count = 0;
    foreach (number; numbers) {
        if ((number % 10) == 0) {

            ++count;
            write(count);

        } else {
            write(' ');
        }

        writeln(": ", number);
    }
}
```

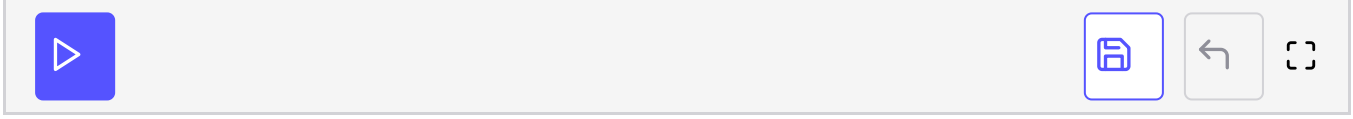


Counter with specific condition

The copy of the element

The `foreach` loop normally provides a copy of the element, not the actual element that is stored in the container. This may cause errors. To see an example of this, let's look at the following program that is trying to double the values of the elements of an array:

```
import std.stdio;

void main() {
    double[] numbers = [ 1.2, 3.4, 5.6 ];

    writefln("Before: %s", numbers);

    foreach (number; numbers) {
        number *= 2;
    }

    writefln("After : %s", numbers);
}
```



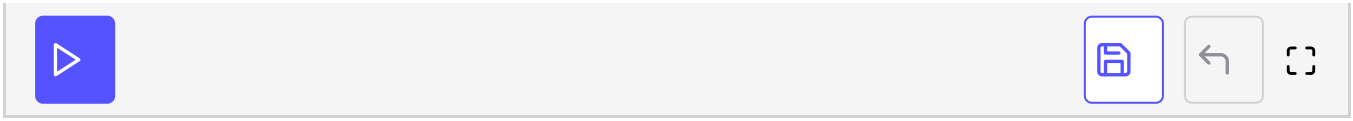
Common cause of bug with foreach loop

The output of the program indicates that the assignment made to each element inside the `foreach` body does not have any effect on the elements of the container.

That is because `number` is not an actual element of the array, but a copy of each element. When the actual elements need to be operated on, the name

must be defined as a reference of the actual element by using the `ref` keyword:

```
import std.stdio;

void main() {
    double[] numbers = [ 1.2, 3.4, 5.6 ];

    writefln("Before: %s", numbers);

    foreach (ref number; numbers) {
        number *= 2;
    }

    writefln("After : %s", numbers);
}
```



Common cause of bug with foreach loop

The new output shows that the assignments now modify the actual elements of the array. The `ref` keyword makes `number` an alias of the actual element at each iteration. As a result, the modifications through number modify the actual elements of the container.

Container integrity

Although it is fine to modify the elements of a container through `ref` variables, the structure of a container must not be changed during iterations. For example, the elements must not be removed or added to the container during a `foreach` loop.
Such modifications may confuse the inner workings of the loop iteration and result in incorrect program states.

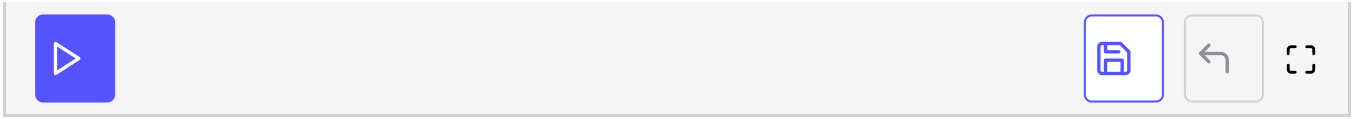
`foreach_reverse`

`foreach_reverse` works the same way as `foreach` except it iterates in the reverse direction:

```
auto container = [ 1, 2, 3 ];
foreach_reverse (element; container) {
```

```
    writefln("%s ", element);
}
```

```
import std.stdio;

void main() {

    auto container = [ 1, 2, 3 ];
    foreach_reverse (element; container) {
        writefln("%s ", element);
    }

}
```



The use of `foreach_reverse` is not common because the range function `retro()` achieves the same goal.

In the next lesson, you will find a coding challenge based on the concepts of `foreach` loop.