Reference Variables

This lesson explains some terminology used with reference variables and the use of the ref keyword.

WE'LL COVER THE FOLLOWING

Terminology

We have frequently used the phrase "provide access to" throughout the course. For example, slices and associative arrays do not own any elements but provide access to elements that are owned by the D runtime. Another phrase, identical in meaning, is "*being a reference of*" as in "*slices are references of zero or more elements*." This is sometimes also used as "*this slice references two elements*." Finally, the act of accessing a value through a reference is called **dereferencing**.

Reference variables are variables that act as aliases of other variables. Although they look like and are used as variables, they do not have values and addresses in memory. Modifications made on a reference variable change the value of the actual variable.

So far, we have already used reference variables in two contexts:

`ref` in `foreach` loops:

The `ref` keyword makes the loop variable the actual element that corresponds to that iteration. When the `ref` keyword is not used, the loop variable is a copy of the actual element.
This can be demonstrated by the `&` operator as well. If their addresses are the same, two variables would be referencing the same value (or the same

element in this case):

```
import std.stdio;

void main() {

    int[] slice = [ 0, 1, 2, 3, 4 ];

    foreach (i, ref element; slice) {
        writeln(&element," : ", &slice[i]);
        assert(&element == &slice[i]);

    }

}
```

Although they are separate variables, the fact that the addresses of `element` and `slice[i]` are the same proves that they have the same value identity. In other words, `element` and `slice[i]` are references of the same value. Modifying either of those affects the actual value. The following memory layout indicates a snapshot of the iteration when i is 3:

Values of the iteration when i = 3

`ref` and `out` function parameters

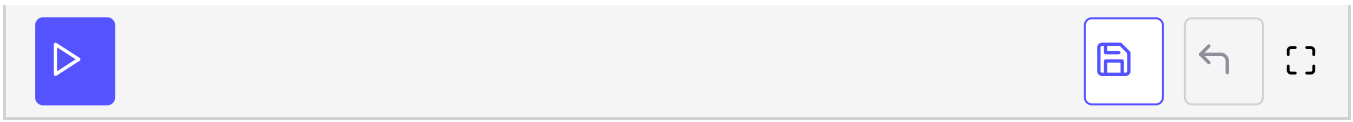
Function parameters that are specified as `ref` or `out` are aliases of the actual variable that the function is called with.
The following example demonstrates this case by passing the same variable to separate `ref` and `out` parameters of a function. Again, the `&` operator indicates that both parameters have the same value identity:

```
import std.stdio;

void main() {
    int originalVariable;
    writeln("address of originalVariable: ", &originalVariable);
```

```
    foo(originalVariable, originalVariable);
}


void foo(ref int refParameter, out int outParameter) {
    writeln("address of refParameter    : ", &refParameter);
    writeln("address of outParameter    : ", &outParameter);
    assert(&refParameter == &outParameter);
}
```



ref and out function parameters

Although they are defined as separate parameters, `refParameter` and `outParameter` are aliases of `originalVariable`.

In the next lesson, we will see reference types.