Arrays

A problem

Often, in our code, we want to store multiple of things. For example, multiple item slots that the player can have:

```
public string item1;
public string item2;
public string item3;
// ...
```

Accessing these items will require a lot of code ->>

```
void UseItem(int number){
  if(number == 1 && item1 != ""){
    Use(item1);
    item1 = "";
  }
  if(number == 2 && item2 != ""){
    Use(item2);
    item2 = "";
  }
  if(number == 3 && item3 != ""){
    Use(item3);
    item3 = "";
  }
  // ...
}
```

Now imagine that for a 100 item slots...

And then for 1000…

How about having 1.000.000 Players?

And what happens, if the number of item slots can vary (depending on the player's level)?

The solution

The solution to this problem are arrays. They allow storing multiple items of the same kind in one collection:

```
// 10 item slots:
string[] items = new string[10];
```

And what's even better: They allow accessing each item using a numeric index:

```
void UseItem(int number){
  if(item[number] != ""){
    Use(item[number]);
    item[number] = "";
  }
}
```

Definition

An **Array** can be defined for **ANY** Type using the syntax:

VARIABLE_TYPE[] variableName;

Initialization

It can be initialized using the syntax:

VARIABLE_TYPE variableName = new VARIABLE_TYPE[length];

e.g.:

int[] numbers = new int[5];

You can even initialize the Array with values by using the Array Initializer Syntax:

```
string[] options = {"Rock", "Paper", "Scissors"};
int[] numbers = {3, 5, 7, 2, 0};
```

Reading and changing

Reading a value is used by accessing the value at index X:

Console.WriteLine(items[2]);

And changing a value at index X:

items[3] = “Spiked Mace”;

Iteration

You can iterate through each item in the array using a for loop and the length of the array:

```
for(int i = 0; i < items.Length; i++){
  Console.WriteLine(items[i]);
}
```

Reference type

At this point, we need to do a short introduction to a rather complex topic: Reference Types. Firstly, let's look at Value Types (which are almost all types that you've used so far):

```
int a = 5; // 5
int b = a; // 5
a = 6;     // 6
Console.WriteLine(a); // 6
Console.WriteLine(b); // 5
```

Here, after assigning one variable a to another b, its value got copied over. Which means, that when assigning a new value to a, b won't be affected by that change.

Now let’s look at an array:

```
int[] a = {5, 5, 5};
int[] b = a;
a[0] = 6;
Console.WriteLine(a[0]); // 6
Console.WriteLine(b[0]); // 6
```

Curious, isn't it? When we assign an Array to a new variable (same, when we pass an Array into a function), then it seems like the Array does not get copied like it does for other Types but instead, changes to one variable of this array affect the other one as well.

We will learn the exact differences between Value and Reference Types later. But for now, make sure to understand that when you hand an Array to a function, then that function can make changes to the contents of the Array and it can affect the calling function.

Detailed examples of using arrays

Creating an Array of 3 Elements:

int[] array = new int[3];

Index	0	1	2
Value	0	0	0

Assigning a value to the Array:

array[1] = 5;

Side note: you cannot assign a value to an array without using the index operator: **array = 5; // error**

Index	0	>1<	2
Value	0	5	0

The first element has the index 0:

array[0] = 3;

Going outside of the size of the array (its bounds) is also not valid:

array[50] = 50; // Error

array[-5] = 10; // Error too, an array always starts at 0

Index	>0<	1	2
Value	3	5	0

Also, trying to assign a string to an integer array results in an error:
array[2] = “Happy Camper”; // error, string is not a valid type for int

Index	0	1	2
Value	3	5	0

You can of course read any value from the array:

Console.WriteLine(array[1]); // prints 5

Index	0	>1<	2
Value	3	5	0

This is also valid: you have created a new Array with a different size, but all elements are empty again:

array = new int[5];

Index	0	1	2	3	4
Value	0	0	0	0	0

You can initialize a new Array with values directly:

array = new int[3] {3, 5, 7};

Index	0	1	2
Value	3	5	7

And this keeps all elements when resizing. Internally, it just copies all the values.

Array.Resize(ref array, 5);

Index	0	1	2	3	4
Value	3	5	7	0	0

You can read the length of an array:

int length = array.Length; // 5

Index	0	1	2	3	4
Value	3	5	7	0	0

You can use the length to do a for loop:

```
for (int i = 0; i < array.Length; i++) {
  Console.WriteLine(array[i]);
}
```

Output:
3
5
7
0
0

Index	0	1	2	3	4
Value	3	5	7	0	0

Goal 1 - Rolling numbers

- Create an array to count the occurrences of random numbers
- roll 10.000 times for a number between 0 and 10
- and count the number of times that you have rolled that specific number
- Afterwards, print the result to the console.

Output:I will roll 10.000 numbers between 0 and 10:
Output:I rolled 0 a total of 987 times.
Output:I rolled 1 a total of 1002 times.
Output:I rolled 2 a total of 998 times.
...

Goal 2 - Looking for numbers

- Ask the user, how many numbers he'd like to input.
- Create an array of that size.
- For each of the array's indices from 0 until Length (exclusive), do:
  - read input from the console
  - cast it to be a number
  - assign it to one of the slots of the array
- Continue asking the user, what number he's looking for.
  - Read the input from the console
  - Cast it to be a number
  - Count, how often that number exists in the array.
    - You can start counting at 0
    - You can iterate over all elements like you did before (when filling it)
    - Then, if the number is the same, you can increase the count by 1
- Print the count to the console