# Arrays of Objects and Primitives

We'll cover the following



The `Array<T>` class represents an array of values in Kotlin. Use arrays only when low-level optimization is desired; otherwise, use other data structures like `List`, which we'll see later in this chapter.

## Creating an array in Kotlin #

The easiest way to create an array is using the `arrayOf()` top-level function. Once you create an array, you may access the elements using the index `[]` operator.

To create an array of `Strings`, for example, pass the desired values to the `arrayOf()` function:

```
val friends = arrayOf("Tintin", "Snowy", "Haddock", "Calculus")

println(friends::class) //class kotlin.Array
println(friends.javaClass) //class [Ljava.lang.String;
println("${friends[0]} and ${friends[1]}") //Tintin and Snowy
```

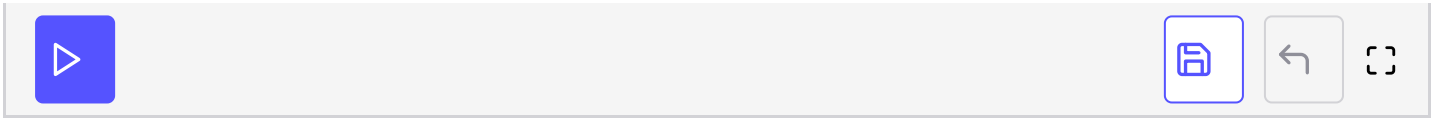
arrays.kts

The `friends` variable holds a reference to the newly created array instance. The type of the object is `Kotlin.Array`, that is `Array<T>`, but the underlying real type, when run on the JVM, is a Java array of `Strings`. To get the values of the elements, the index operator `[]` is used and that, in turn, will invoke the `get()` method of `Array<T>`. When used on the left-hand side, the index operator `[]` will invoke the `set()` method of `Array<T>`.

The previous code created an array of `Strings`; to create an array of integers we may be tempted to use the same method, as in this example:

```
val numbers = arrayOf(1, 2, 3)

println(numbers::class) //class kotlin.Array
println(numbers.javaClass) //class [Ljava.lang.Integer;
```



arrays.kts

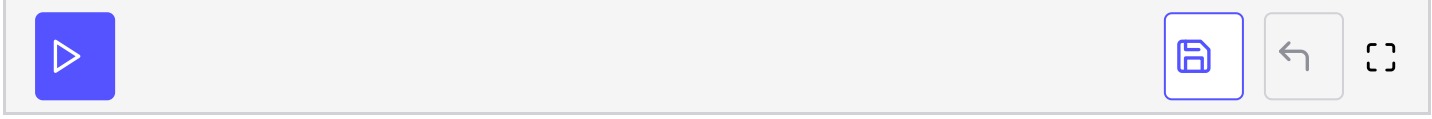
That works, but it may not be a smart way. When numbers were passed to `arrayOf()`, the instance created was `Array<T>`, as expected, but internally it is an array of boxed `Integer` types. That overhead is unnecessary when working with primitive types.

### Specialized arrays #

The specialized functions like `intArrayOf()` are better alternatives to create specialized arrays that don't have the boxing overhead. To create an array of `ints` instead of an array of `Integers`, change the previous code to the following:

```
val numbers = intArrayOf(1, 2, 3)

println(numbers::class) //class kotlin.IntArray
println(numbers.javaClass) //class [I
```



arrays.kts

The same operations that are available on `Array<T>` are available on the specialized classes like `IntArray` as well. So even though they are different types, you may use them as if they were the same type.

## More functionalities of arrays #

In addition to using the index operator `[]` to get and set values, you may determine the size of the array using the `size` property. Also, you may use one of many functions on Array to conveniently work with arrays. Let's exercise the `size`

property and one useful method— `average` —on the array we just created:

```
// arrays.kts
println(numbers.size) //3
println(numbers.average()) //2.0
```

Explore the `Kotlin.Array<T>` class to learn about the different methods you may use on an array of objects and primitives.

Instead of hard-coding the values when creating an array, you may also compute the values if you like. For example, in the code that follows we're computing the square of values from *1 to 5* into an array and then totaling the values in the array:

```
// arrays.kts
println(Array(5) { i -> (i + 1) * (i + 1) }.sum()) //55
```

The Array constructor takes two parameters, the size of the array and a function that takes the index, starting with `0`, and returns the value to be placed at that index. The syntax for this function in the example uses a lambda expression, which we'll explore further in Chapter 11, Functional Programming with Lambdas.

QUIZ

1 How do you create and access arrays in Kotlin?

2

What is the output of the following code snippet?

```
println(Array(5) { i -> (i + 1) * (i + 1) }.count())
```

Retake Quiz

If you want an ordered collection of elements with flexible size, then you may want to consider a list instead of an array. Also, unlike arrays, which are mutable, lists come in both mutable and immutable flavors which we'll cover in the next lesson.