Arrays

The runtime system provides only three data types: the primitive number types (including `Char` and `Boolean`), classes with a fixed number of fields (attributes), and arrays.

Everything else needs to be implemented using these basic building blocks. For instance, `String` stores characters in an array. `List` is implemented using an array.

An array is simply a block of memory of user-defined length that stores references to objects of some type.
You can imagine an array as a mutable list of fixed length.

Using Arrays

We usually use `List` or `MutableList` instead of an array.

However, the `args` variable in Kotlin scripts (which stores the command line arguments), is of type `Array<String>`.

```
>>> val a = arrayOf(1, 2, 3)
>>> val l = listOf(1, 2, 3)
>>> println(l)
[1, 2, 3]
>>> println(a)
[Ljava.lang.Integer;6a1aab78
>>> val b = arrayOf(1, 2, 3)
>>> val m = listOf(1, 2, 3)
>>> l == m
true
>>> a == b
false
```

Beware, arrays are primitive!

no nice `toString()` method!

equality operator does not look at the contents of arrays

Creating arrays

We can create short arrays by listing the elements: `arrayOf(1, 2, 3, 4)`.

When the number of elements is large, or not known in advance, you have to do it differently.

An array of 100 zeroes:

```
>>> val zeros = Array(100) { 0 }
```

code computing the value

Can use the magic variable `it` (the index of each element):

```
>>> val squares = Array(10) { it * it }
>>> squares.joinToString(" ")
0 1 4 9 16 25 36 49 64 81
```

Or simply use a `MutableList` instead.

Two-dimensional arrays

In many applications we need a two-dimensional table or matrix of elements.

We use two indices, usually called row and column.

We use one array for each row. Its elements are the cells of this row, one for each column.

Then we use one array that stores all the row arrays.

Creating m rows of n columns:

```
>>> val t = Array(m) { Array(n) { 0 } }
```

The type of `t` is `Array<Array<Int>>`.

(We could do this with `List`, but it can use much more memory.)

Working with two-dimensional array

Accessing the elements:

```
>>> t[2][4] = 13
>>> t[0][0] = -3
>>> t[1][5] = 99
```

Checking the dimensions:

```
>>> t.size   // #rows
3
>>> t[0].size // #columns
6
```

Printing the matrix (you are on your own):

```
>>> t.joinToString(separator="\n",
                   transform={it.joinToString()})
-3, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 99
0, 0, 0, 0, 13, 0
```