Arrays

Donald Pinckney and Ash Dreyer

Table of Contents

Which Problem do Arrays Solve?

Many Constants / Variables

- Often we need to have a lot of constants or variables
- Sometimes, the number of constants or variables depends on user input or other conditions
- Arrays are a collection of values
- Arrays can have any number of values in them

Example

```
  1> let array1: [Int] = [0, 1, 4, 9, 16]
array1: [Int] = 5 values {
  [0] = 0
  [1] = 1
  [2] = 4
  [3] = 9
  [4] = 16
}
```

Type of an Array

- You already know: `Int, Double, String, Bool`
- For **ANY TYPE**, you can make an array of that type:
  - `[Int]`
  - `[Double]`
  - `[String]`
  - `[[Double]]`

Creating Arrays

Creating Arrays - 3 Common Ways

- Create with a list of values you already know:
  - `let array1 = [1, 8, 4, x, y, 7] // x and y are already variables`
- Create an empty array (you can add values later):
  - `var array2: [Int] = []`
- Create an array of a repeating value:
  - `let array3 = [Int](repeating: 0, count: 100)`

Examples

```
  1> let x = 5
x: Int = 5
  2> let y = 9
y: Int = 9
  3> let array1 = [1, x, 6, 5, y]
array1: [Int] = 5 values {
  [0] = 1
  [1] = 5
  [2] = 6
  [3] = 5
  [4] = 9
}
```

Examples

```
  4> let array2: [String] = []
array2: [String] = 0 values
  5> let array3 = [Double](repeating: 0, count: 100)
array3: [Double] = 100 values {
  [0] = 0
  [1] = 0
   …
  [98] = 0
  [99] = 0
}
```

Accessing Arrays

Accessing Arrays

- Each value in the array is numbered, from 0 to `count - 1`
- This numbering is called an **index,** and it must be an `Int`

```
  1> let array1 = [5, 7, 4]
array1: [Int] = 3 values {
  [0] = 5
  [1] = 7
  [2] = 4
}
```

Accessing at an Index

```
  1> let array1 = [5, 7, 4]
array1: [Int] = 3 values {
  [0] = 5
  [1] = 7
  [2] = 4
}
  2> print(array1[0])
5
  3> print(array1[1])
7
  4> print(array1[2])
4
```

How Long is an Array?

- **Very** often you want to know how many values are in an array

```
  1> let array1 = [5, 7, 4]
array1: [Int] = 3 values {
  [0] = 5
  [1] = 7
  [2] = 4
}
  2> print(array1.count)
3
```

Indexing + Count Example: Last Value

```
  1> let array1 = [5, 7, 4]
array1: [Int] = 3 values {
  [0] = 5
  [1] = 7
  [2] = 4
}
  2> print(array1[array1.count - 1])
4
```

Modifying Arrays

Variables

- If you are going to modify an array, you **need** to declare it as `var`, instead of `let`.

Modifying Arrays - 2 Common Ways

- Put a new value in the location of an index
  - `array1[2] = 93`
  - Make sure the array is long enough that this index exists
- Make the array longer by adding a value to the end
  - `array1.append(82)`

Example

```
  1> var array1 = [5, 7, 4]
array1: [Int] = 3 values {
  [0] = 5
  [1] = 7
  [2] = 4
}
  2> array1[array1.count - 1] = 10
  3> print(array1)
[5, 7, 10]
  4> array1.append(16)
  5> print(array1)
[5, 7, 10, 16]
```

Iterating Over Arrays

What is Iterating, and Why?

- Iterating over arrays means to consecutively access each value in an array
- This is **super** common: almost all interesting array calculations do this.
  - Example: find the largest `Int` in an array
  - Access each value in the array, and see if it is larger than the current largest

How Would *You* Iterate?

How to Iterate?

- You already have all the tools to iterate
- A while loop lets you **repeat** code, and you want to repeat some code for every valid index.
- The valid indices start at 0 and end at and include `count - 1`
- The boolean condition in the while loop lets you filter to only valid indices
- In an earlier slide we see how to get the `count` with `myArray.count`

Iterating With While Loops

```
  1> let array1 = [5, 7, 4]
array1: [Int] = 3 values {
  [0] = 5
  [1] = 7
  [2] = 4
}
  2> var i = 0
i: Int = 0
  3> while i < array1.count {
  4.     print(array1[i])
  5.     i += 1
  6. }
5
7
4
```