Arreglos

Un arreglo es una colección de datos que comparten un contexto y tienen el mismo tipo de dato.

Se pueden declarar y asignar valores directamente, no es necesario indicar el tipo de dato pero Swift le asignará uno automáticamente de acuerdo a los datos que se almacenan.

```
let arreglo = [1, 3, 2, 5]
print("Arreglo = \(arreglo)")
```

Para indicar explícitamente el tipo de dato que se desea asignar a los datos que contedrá el arreglo, se debe indicar ese tipo de dato deseado al declararse y escribirse entre corchetes.

```
let arreglo:[Int] = [1, 3, 2, 5]
print("Arreglo = \(arreglo)")
```

Arreglos multidimensionales

Si se desea un arreglo de varias dimensiones, se puede crear directamente asignando los valores. Cada dimensión debe tener sus valores entre corchetes, y las dimensiones deben separarse con comas. En swift, no es necesario que tenga la misma cantidad de elementos por dimensión.

```
let arreglo = [
    [2, 3, 5, 7, 11, 13],
    [1, 1, 2, 3, 5, 8, 13, 21],
    [1, 4, 9, 16, 25],
]
print("Arreglo = \(arreglo)")
```

Acceso y modificación de datos

Se puede tener acceso a los datos de un arreglo a través de los métodos heredados de la alase `Array` o mediante la sintaxis subscript.

Para determinar el número de elementos del arreglo, use la propiedad `count`.

```
let arreglo1:[Int] = [1, 3, 2, 5]
print("Arreglo 1 dimensión= \(arreglo1)")
print("Numero de elementos: \(arreglo1.count)")
```

Si el arreglo es multidimensional, la propiedad `count` mostrará la cantidad de elementos de la primer dimensión (la más exterior o inmediata).

```
let arreglo2 = [
    [2, 3, 5, 7, 11, 13],
    [1, 1, 2, 3, 5, 8, 13, 21],
    [1, 4, 9, 16, 25],
]
print("Arreglo 2 dimensiones = \(arreglo2)")
print("Numero de elementos: \(arreglo2.count)")

let arreglo3 = [
[
    [2, 3, 5, 7, 11, 13],
    [1, 1, 2, 3, 5, 8, 13, 21],
    [1, 4, 9, 16, 25],
],
[
    [2, 3, 5, 7, 11, 13],
    [1, 1, 2, 3, 5, 8, 13, 21],
    [1, 4, 9, 16, 25],
]
]
print("Arreglo 3 dimensiones = \(arreglo3)")
print("Numero de elementos: \(arreglo3.count)")
```

Se puede utilizar la propiedad booleana `isEmpty` para determinar si el arreglo esta vacio o no.

```
var shoppingList = ["Eggs", "Milk"]
print("The shopping list contains \(shoppingList.count) items.")
if shoppingList.isEmpty {
    print("The shopping list is empty.")
} else {
    print("The shopping list isn't empty.")
}
```

Se pueden añadir elementos al final del arreglo con el método heredado `append`.

```
shoppingList.append("Flour")
```

Además, se pueden agregar uno o más elementos compatibles al final del arreglo con el operador `+=`.

```
shoppingList += ["Baking Powder"]
// shoppingList now contains 4 items

shoppingList += ["Chocolate Spread", "Cheese", "Butter"]
// shoppingList now contains 7 items
```

Para obtener un valor contenido por el arreglo, se puede utilizar la sintáxis subscript para localizarlo. Considere que el conteo de elementos comienza en 0.

```
var firstItem = shoppingList[0]
// firstItem is equal to "Eggs"
```

Se puede utilizar la misma sintaxis para cambiar un valor específico en el arreglo.

```
shoppingList[0] = "Six eggs"
// the first item in the list is now equal to "Six eggs" rather than "Eggs"
```

También se puede utilizar la sintáxis subscript para cambiar un rango de valores simultáneamente, incluso si el conjunto de valores de reemplazo tiene un tamaño diferente al rango que será reemplazado. El siguiente ejemplo reemplaza “Chocolate Spread”, “Cheese”, y “Butter” con “Bananas” y “Apples”:

```
shoppingList[4...6] = ["Bananas", "Apples"]
// shoppingList now contains 6 items
```

Para insertar un elemento al arreglo en un índice específico, se puede usar el método `insert(_:at:)`.

```
shoppingList.insert("Maple Syrup", at: 0)
// shoppingList now contains 7 items
// "Maple Syrup" is now the first item in the list
```

Esto agrega el elemento `"Maple Syrup"` al inicio del arreglo, en la posción 0.

De manera similar, se puede eliminar un elemento con el método `remove(at:)`. Este método eliminar el elemento seleccionado por el índice y lo devuelve.

```
let mapleSyrup = shoppingList.remove(at: 0)
// the item that was at index 0 has just been removed
// shoppingList now contains 6 items, and no Maple Syrup
// the mapleSyrup constant is now equal to the removed "Maple Syrup" string
```

Cualquier hueco que quede en el arreglo es rellenado desplazando los elementos posteriores. En el ejemplo, los elementos se desplazaron y ahora en la posición 0 se encuentra `"Six eggs"`.

Iterando un arreglo

Se puede iterar los valores de un arreglo con el ciclo `for-in`.

```
for item in shoppingList {
```

```
    print(item)
}
// Six eggs
// Milk
// Flour
// Baking Powder
// Bananas
```

Si necesita el índice de cada elemento así como su valor, utilice el método `enumerated()` para iterar sobre todo el arreglo. Para cada elemento del arreglo, el método `enumerated()` devuelve una tupla compuesta de un entero y el elemento. Se puede descomponer la tupla en constantes temporales o variables dentro del ciclo.

```
for (index, value) in shoppingList.enumerated() {
    print("Item \(index + 1): \(value)")
}
// Item 1: Six eggs
// Item 2: Milk
// Item 3: Flour
// Item 4: Baking Powder
// Item 5: Bananas
```

```
for (var indice, var valor) in arreglo1.enumerated() {
    print("Item \(indice + 1): \(valor)")
}
```

Crear un arreglo vacío Si se desea crear un arreglo vacío que posteriormente se llenará en el flujo del programa, se puede hacer en la declaración del mismo, sin especificar los valores.

```
let x:[Int]
```

Si desea un arreglo de dos o más dimensiones, debe indicarlo en la declaración con la misma cantidad de pares de corchetes como de dimensiones que desea para el arreglo.

```
let x:[[String]]
```

Crear un arreglo no vacío Si por otro lado, necesita crear un arreglo y rellenarlo con algún valor específico, puede hacerlo mediante la clase `Array` y los parámetros `repeating` y `count`.

```
Array(repeating: 3, count: 5)
```

Este código genera un arreglo con 5 elementos 3.

Si se desea rellenar con números enteros aleatorios, se puede utilizar el método `random` de la clase

`Int`.

```
Array(repeating: Int.random(in: 0..<20), count: 5)
```

Esta instrucción genera un arreglo con un número aleatorio entre 0 y 20 repetido 5 veces.

El siguiente código crea un arreglo de tamaño 10 y lleno con ceros de tipo `Float`.

```
Array(repeating: Float(), count: 10)
```

Puede cambiar Float por algún otro tipo de dato numérico si así lo requiere. Si utiliza otro tipo de dato no numérico, el arreglo se llenará con el valor por defecto del tipo de dato que corresponda.

Si requiere un arreglo de dos dimensiones o más, deberá anidar la instrucción anterior tantas veces como lo requiera.

```
Array(repeating: Array(repeating: 3, count: 5), count: 4)
```

Referencias

https://docs.swift.org/swift-book/LanguageGuide/CollectionTypes.html

https://www.programiz.com/swift-programming/operators#other

https://stackoverflow.com/questions/25127700/two-dimensional-array-in-swift

https://www.zerotoappstore.com/how-to-use-a-two-d-array.html

Anterior | **Inicio** | **Siguiente**