Swift 6

Colecciones

Estructuras de datos

En programación, una estructura de datos es una forma particular de organizar datos en una computadora para que pueda ser utilizado de manera eficiente.

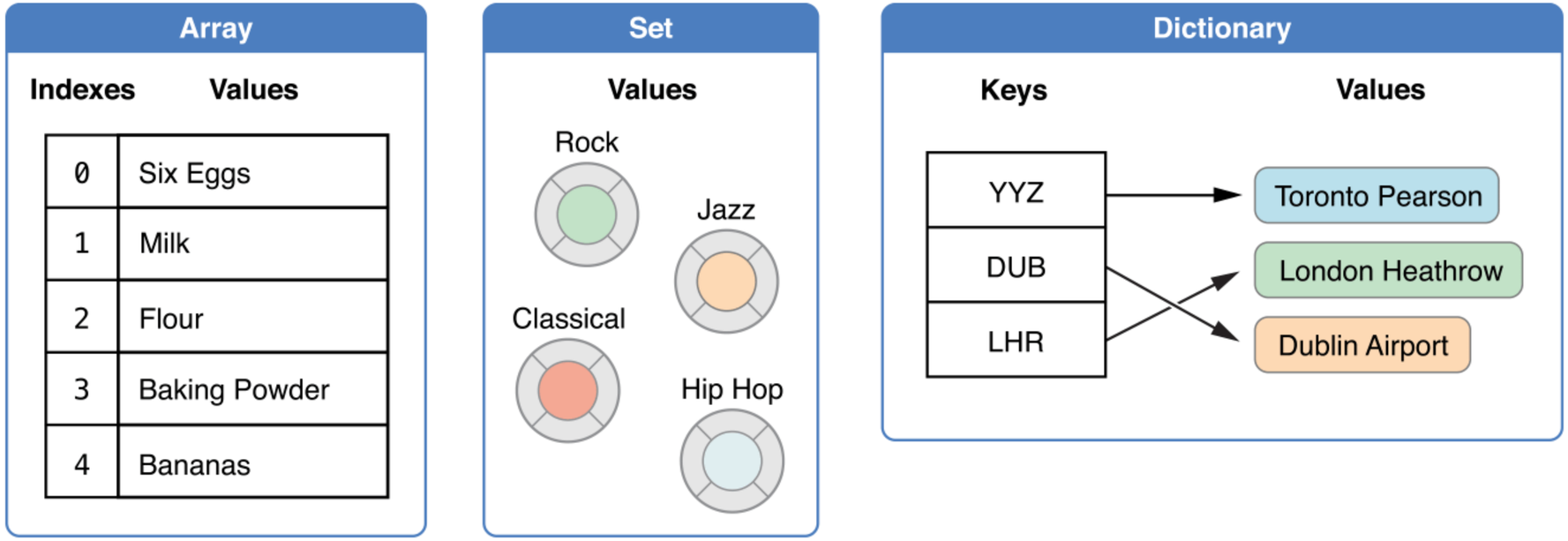
Diferentes tipos de estructuras de datos son adecuados para diferentes tipos de aplicaciones, y algunos son altamente especializados para tareas específicas.

https://es.wikipedia.org/wiki/Estructura_de_datos

Estructuras de datos

- Organización de datos
- Operaciones posibles

Colecciones en Swift



Arrays

En programación, una matriz o vector (llamado en inglés array) es una zona de almacenamiento continuo, que contiene una serie de elementos del mismo tipo, los elementos de la matriz.

https://es.wikipedia.org/wiki/Vector_(informática)

Arrays

```
var otherInts = Array<Int>()

var someInts = [Int]()  // Sintaxis preferida

someInts.append(3)

someInts = []

var threeDoubles = Array(repeating: 0.0, count: 3)

var anotherThreeDoubles = Array(repeating: 2.5, count: 3)

var sixDoubles = threeDoubles + anotherThreeDoubles
```

Arrays

```
var shoppingList: [String] = ["Eggs", "Milk"]

var firstItem = shoppingList[0]

shoppingList[0] = "Six eggs"

var cinemaShoppingList = ["Chocolate", "Popcorn"]
```

Características de los arrays

- Se numeran desde 0.
- Usan la inferencia de tipo si los inicializamos al crearlos
- El acceso a una posición inexistente provoca un error de tiempo de ejecución
- Si se declara con **let** no puede variar el número de elementos ni el contenido

Operaciones sobre arrays

- Se puede preguntar cuantos elementos hay con **.count**
- Se puede preguntar si está vacío con **.isEmpty**

Operaciones sobre arrays

```
print("The shopping list contains \(shoppingList.count) items.")


if shoppingList.isEmpty {
    print("The shopping list is empty.")
} else {
    print("The shopping list is not empty.")
}
```

Operaciones sobre arrays

- Se pueden añadir elementos con **.append(_:)**
- Se pueden concatenar arrays con el operador **+=**
- Se pueden insertar elementos con **.insert(_:at:)**
- Se puede eliminar elementos con **.remove(at:)**
- Se puede eliminar el último elemento con **.removeLast()**

Operaciones sobre arrays

```
shoppingList.append(“Flour")

shoppingList += ["Baking Powder"]

shoppingList += ["Chocolate Spread", "Cheese", "Butter"]

shoppingList[4...6] = ["Bananas", "Apples"]

shoppingList.insert("Maple Syrup", at: 0)

let mapleSyrup = shoppingList.remove(at: 0)

let apples = shoppingList.removeLast()
```

Recorrer un array

```
for item in shoppingList {
    print(item)
}
```

Recorrer un array

```
for (index, value) in shoppingList.enumerated() {
    print("Item \(index + 1): \(value)")
}
```

Crea un array de nombres y realiza las siguientes operaciones:

1. Agrega dos nombres nuevos.
2. Elimina el tercer nombre.
3. Imprime el total de nombres.

Conjuntos

En informática, un conjunto es una colección (contenedor) de ciertos valores, sin ningún orden concreto ni valores repetidos.

https://es.wikipedia.org/wiki/Conjunto_(programación)

Conjuntos

- Almacenan elementos del mismo tipo
- No hay orden definido
- Se usa cuando no importa el orden o necesitamos que los elementos no se repitan
- Los valores tienen que ser “*hashable*”
- Si se declara con **let** no puede variar el número de elementos ni el contenido una vez inicializado

Hashable

- Valor numérico único para un objeto concreto
- Permite comparar objetos entre sí
- Los tipos básicos de Swift son *hashable*
- Si queremos que nuestros propios tipos lo sean tienen que implementar el protocolo **Hashable**

Conjuntos

```
var letters = Set<Character>()

print("letters is of type Set<Character> with \(letters.count) items.")

letters.insert("a")

letters = []


var favoriteGenres: Set<String> = ["Rock", "Classical", "Hip hop"]

var myFavoriteGenres: Set = ["Rock", "Classical", "Hip hop"]
```

Operaciones sobre conjuntos

- Se puede preguntar cuantos elementos hay con **.count**
- Se puede preguntar si está vacío con **.isEmpty**
- Se pueden añadir elementos con **.insert(_:)**
- Se pueden eliminar elementos con **.remove(_:)** o **.removeAll()**
- Se puede consultar si un elemento existe con **.contains(_:)**

Operaciones sobre conjuntos

```
print("I have \(favoriteGenres.count) favorite music genres.")


if favoriteGenres.isEmpty {
    print("As far as music goes, I'm not picky.")
} else {
    print("I have particular music preferences.")
}


favoriteGenres.insert("Jazz")
```

Operaciones sobre conjuntos

```
if let removedGenre = favoriteGenres.remove("Rock") {
    print("\(removedGenre)? I'm over it.")
} else {
    print("I never much cared for that.")
}
```

Operaciones sobre conjuntos

```
if favoriteGenres.contains("Funk") {
    print("I get up on the good foot.")
} else {
    print("It's too funky in here.")
}
```

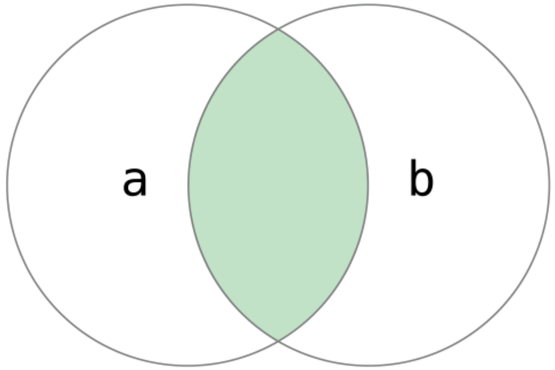
Recorrer un conjunto

```
for genre in favoriteGenres {
    print("\(genre)")
}
```

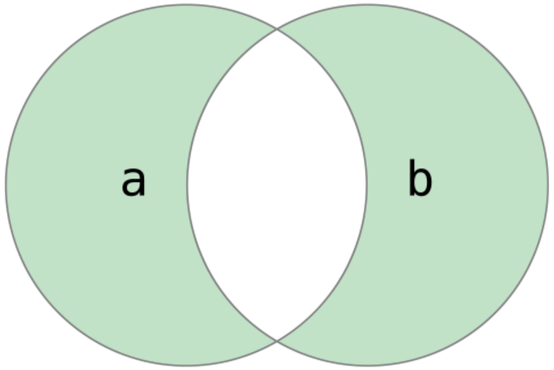
```
for genre in favoriteGenres.sorted() {
    print("\(genre)")
}
```

Operaciones entre conjuntos

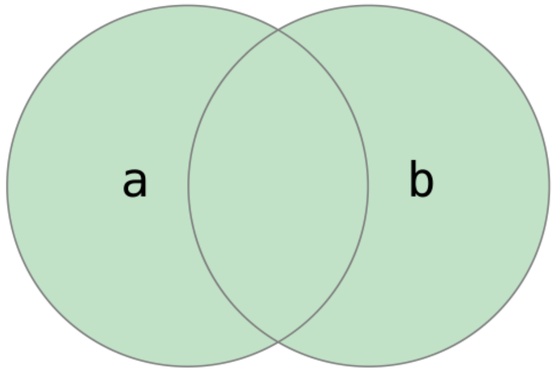
a.intersection(b)



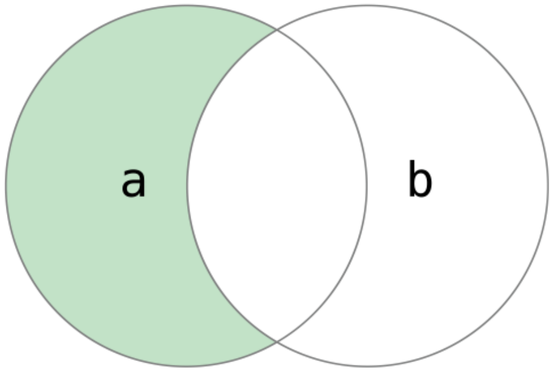
a.symmetricDifference(b)



a.union(b)



a.subtracting(b)



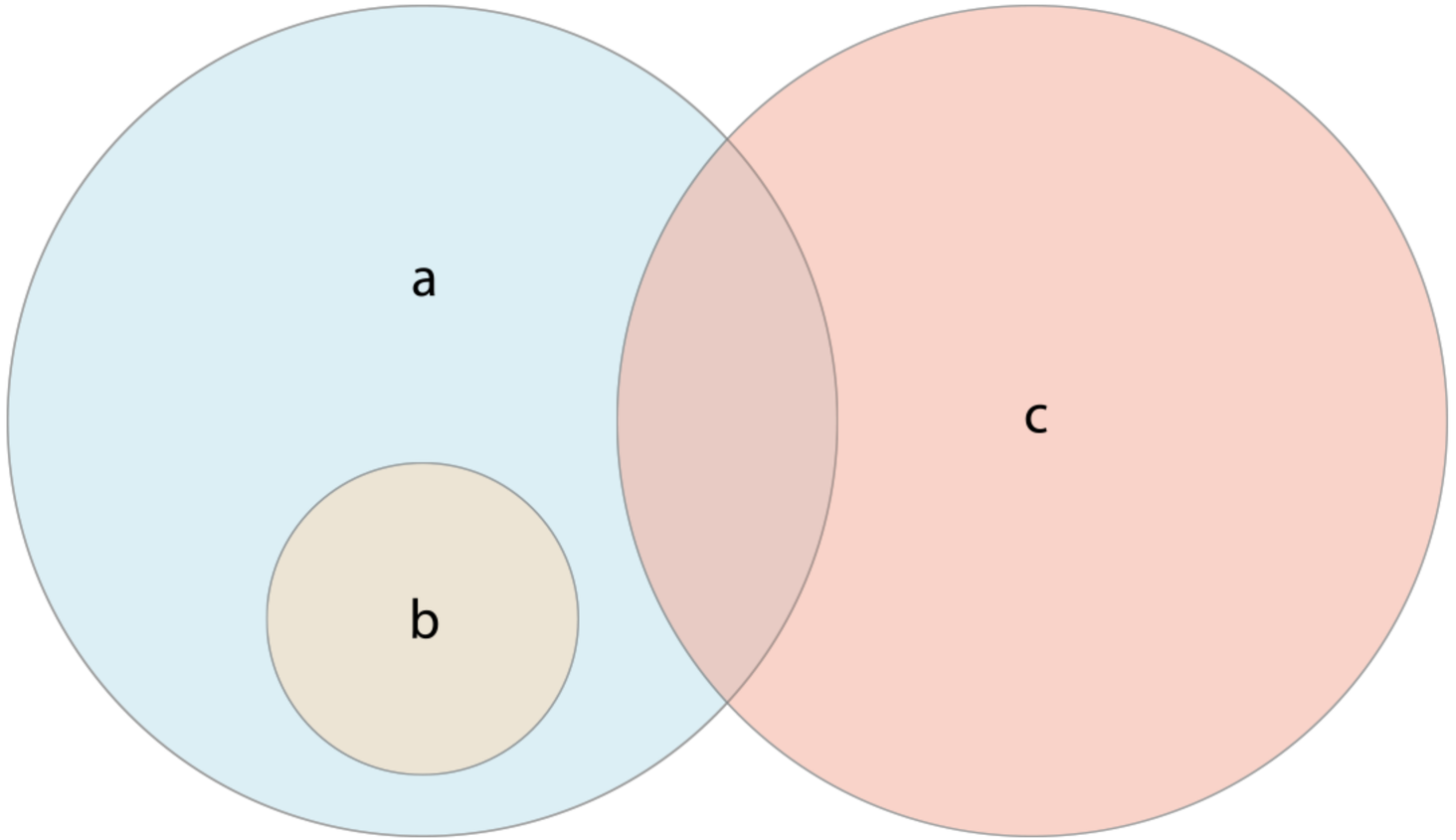
Operaciones entre conjuntos

- La operación **intersection(_:)** crea un nuevo conjunto que contiene sólo los valores comunes a los dos conjuntos
- La operación **symmetricDifference(_:)** crea un nuevo conjunto con los valores que no sean comunes a los dos conjuntos
- La operación **union(_:)** crea un nuevo conjunto con los valores comunes a los dos conjuntos
- La operación **subtracting(_:)** crea un nuevo conjunto con los valores que no estén en el conjunto especificado

Operaciones entre conjuntos

```
let oddDigits: Set = [1, 3, 5, 7, 9]
let evenDigits: Set = [0, 2, 4, 6, 8]
let singleDigitPrimeNumbers: Set = [2, 3, 5, 7]

oddDigits.union(evenDigits).sorted()
// [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

oddDigits.intersection(evenDigits).sorted()
// []

oddDigits.subtracting(singleDigitPrimeNumbers).sorted()
// [1, 9]

oddDigits.symmetricDifference(singleDigitPrimeNumbers).sorted()
// [1, 2, 9]
```

Pertenencia e igualdad de conjuntos



Pertenencia e igualdad de conjuntos

- El operador == permite comprobar si dos conjuntos contienen los mismos valores
- La operación **isSubset(of:)** determina si todos los valores de un conjunto están contenidos en otro
- La operación **isSuperset(of:)** determina si un conjunto contiene todos los valores de otro
- Las operaciones **isStrictSubset(of:)** o **isStrictSuperset(of:)** determinan si un conjunto es un subconjunto o superconjunto, pero no igual, a un conjunto dado
- La operación **isDisjoint(with:)** determina si dos conjuntos tienen algún valor el común

Pertenencia e igualdad de conjuntos

```
let houseAnimals: Set = ["🐶", "🐱"]
let farmAnimals: Set = ["🐮", "🐔", "🐑", "🐶", "🐱"]
let cityAnimals: Set = ["🐳", "🐭"]

houseAnimals.isSubset(of: farmAnimals)
// true

farmAnimals.isSuperset(of: houseAnimals)
// true

farmAnimals.isDisjoint(with: cityAnimals)
// true
```

Diccionarios

Diccionarios

```
var namesOfIntegers = [Int: String]()

namesOfIntegers[16] = "sixteen"

namesOfIntegers = [:]
```

Diccionarios

```
var airports: [String: String] = ["TYO": "Tokyo", "DUB": "Dublin"]

var moreAirports = ["TYO": "Tokyo", "DUB": "Dublin"]
```

Diccionarios

```
print("The dictionary of airports contains \(airports.count) items.")


if airports.isEmpty {
    print("The airports dictionary is empty.")
} else {
    print("The airports dictionary is not empty.")
}
```

Diccionarios

```
airports["LHR"] = "London"  // Añadir un elemento

airports["LHR"] = "London Heathrow"  // Actualizar el elemento

airports["APL"] = "Apple International"
airports["APL"] = nil  // Borrar un elemento
```

Características de los diccionarios

- Almacenan parejas de elementos clave-valor (*key-value*)
- El tipo de la clave tiene que ser "*hashable*" (los tipos básicos lo son)
- Si se declara con **let** no puede variar el número de elementos ni el contenido una vez inicializado

Operaciones sobre diccionarios

- Se puede preguntar cuantos elementos hay con **.count**
- Se puede preguntar si está vacío con **.isEmpty**
- Se puede modificar un valor con **.updateValue(_:forKey:)** que devuelve el valor antiguo como un opcional (la sintaxis con **[]**, no)
- Se puede eliminar un valor con **.removeValue(forKey:)** que devuelve el valor antiguo como un opcional (la sintaxis con **[]=nil**, no)

Operaciones sobre diccionarios

```
if let oldValue = airports.updateValue("Dublin Airport", forKey: "DUB") {
    print("The old value for DUB was \(oldValue).")
}
```

```
if let airportName = airports["DUB"] {
    print("The name of the airport is \(airportName).")
} else {
    print("That airport is not in the airports dictionary.")
}
```

Operaciones sobre diccionarios

```
if let removedValue = airports.removeValue(forKey: "DUB") {
    print("The removed airport's name is \(removedValue).")
} else {
    print("The airports dictionary does not contain a value for DUB.")
}
```

Recorrer un diccionario

```
for (airportCode, airportName) in airports {
    print("\(airportCode): \(airportName)")
}
```

Recorrer un diccionario

```
for airportCode in airports.keys {
    print("Airport code: \(airportCode)")
}
```

```
for airportName in airports.values {
    print("Airport name: \(airportName)")
}
```

Extraer los valores a arrays

```
let airportCodes = [String](airports.keys)

let airportNames = [String](airports.values)
```

Crea un diccionario para registrar las calificaciones de alumnos. Luego:

1. Agrega las calificaciones de 3 alumnos.
2. Modifica la calificación de uno de ellos.
3. Elimina un alumno.
4. Calcula la media de las calificaciones.

Tuplas

Tipo de datos compuesto, que si bien no son exactamente una estructura de datos como los arrays o los diccionarios, pero son muy útiles para agrupar múltiples valores en un solo conjunto, especialmente cuando esos valores pueden ser de diferentes tipos.

Tuplas

1. Agrupan múltiples valores en una sola unidad.
2. Los valores pueden ser de diferentes tipos.
3. Inmutables por defecto, pero se pueden declarar como mutables usando *var*.

Tuplas

```
//Declaración básica

Let person = ("John", 30, true)

print(person.0) //John

print(person.1) //John

print(person.2) //true
```

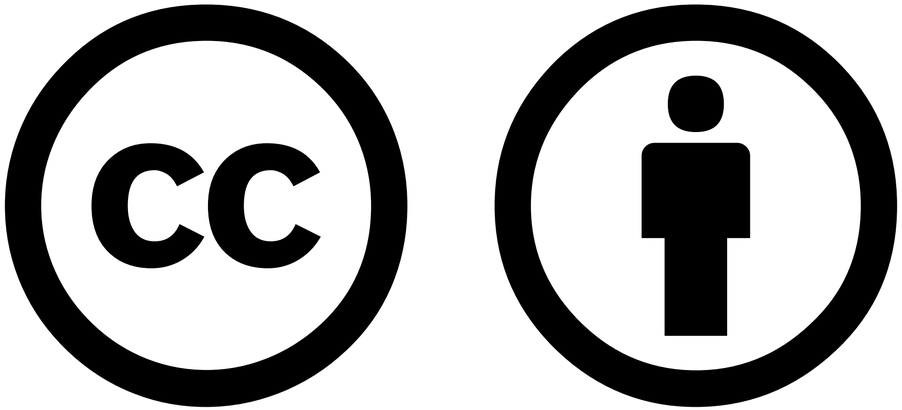
Tuplas

```
//Declaración con nombres de elementos

let  user = (name: "Alice", age: 25, isActive: true)

print(user.name) // "Alice"

print(user.age) // 25

print(user.isActive) // true
```

Tuplas

Uso común: retornar múltiples valores de una función

```
func getUserInfo() -> (name: String, age: Int) {
    return ("Bob", 28)
}
let userInfo = getUserInfo()
print(userInfo.name) // "Bob"
print(userInfo.age)  // 28
```



Excepto si se especifica lo contrario, esta presentación está bajo licencia

https://creativecommons.org/licenses/by/4.0/

© 2017 Ion Jaureguialzo Sarasola. Algunos derechos reservados.
© 2024 Inés Larrañaga Fdez. De Pinedo. Algunos derechos reservados.