
SIGMA
Software


SIGNL
Software
UNIVERSITY


SIGMA
Software
UNIVERSITY

ЛЕКЦІЯ 4

“Колекції”

4

SIGMA SOFTWARE
UNIVERSITY

Колекції

Колекції

Масиви

Множини

Словники

Інші колекції

SIGMA SOFTWARE UNIVERSITY

Колекції

Колекції - це узагальнена назва сутності, яка є контейнером для інших сутностей, що мають деякі спільні властивості та операції над якими мають виконуватися в однаковий спосіб. У загальному, виділяють наступні типи колекцій:

Спи́ски (lists)
Множи́ни (sets)
Мультимножи́ни (multisets)
Дере́ва (trees)
Гра́фи (graphs)
Сте́ки (stacks)
Черги́ (queues)
Асоціативні масиви *або* словники́ (dictionaries)

Колекції

OBJ-C

Objective-C надає вбудовану реалізацію наступних типів колекцій:

NSArray, NSMutableArray (список *або* масив)
NSSet, NSMutableSet, NSOrderedSet, NSCountedSet (множина)
NSDictionary, NSMutableDictionary (асоціативний масив *або* словник)

CF - альтернативи
NSPointerArray
NSHashTable
NSMapTable

*можуть містити будь-які типи обєктів

SIGMA SOFTWARE UNIVERSITY

Колекції

Swift надає вбудовану реалізацію наступних типів колекцій:

Array (список *або* масив)
Set (множина)
Dictionary (асоціативний масив *або* словник)

ArraySlice (зріз масиву)
CollectionOfOne (колекція з одного елемента)
ContinousArray (послідовний масив)
EmptyCollection (пуста колекція)
Range (діапазон або проміжок)

Колекції

Колекції

Масиви

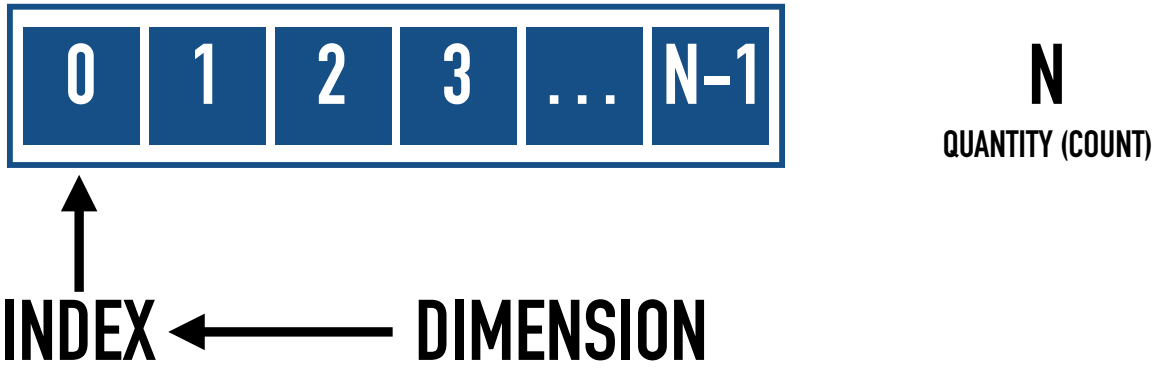
Множини

Словники

Інші колекції

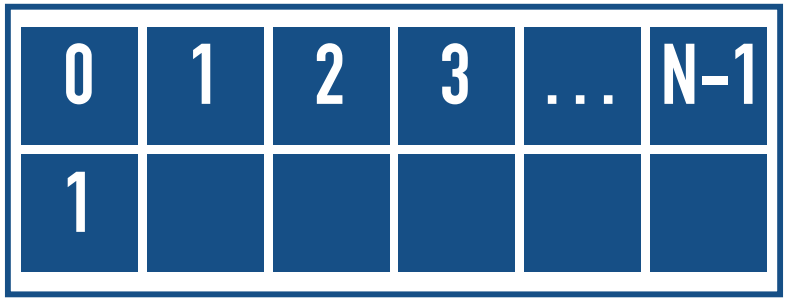
Масиви

Массив - впорядкований набір фіксованої кількості *однотипних* елементів, що зберігаються в *послідовно розташованих* комірках оперативної пам'яті, мають порядковий номер і спільне ім'я, що надає користувач.



Масиви

Массив - впорядкований набір фіксованої кількості *однотипних* елементів, що зберігаються в *послідовно розташованих* комірках оперативної пам'яті, мають порядковий номер і спільне ім'я, що надає користувач.



Масиви

Swift визначає масив, як тип за значенням.

SIGMA SOFTWARE UNIVERSITY

Масиви

OBJ-C

Objective-C надає підримку вбудованого класу NSArray та NSMutableArray
Формат опису масиву на Objective-C має наступний вигляд:

```
NSArray<Type> *arrayName = @[ value1, value2 …valueN];
NSMutableArray<Type> *array = [NSMutableArray arrayWithObjects:obj1, obj2…nil];
```

Масиви

Swift надає підримку вбудованого класу Array, що представляє масив.
Формат опису масиву на Swift має наступний вигляд:

```
let arrayName: Array<Array<…Array<Type>…>> = arrayValue
var arrayName: Array<Array<…Array<Type>…>> = arrayValue
```

Масиви

Swift надає підримку вбудованого класу Array, що представляє масив.
Формат опису масиву на Swift має наступний вигляд:

```
let arrayName: [[…[Type]…]] = arrayValue
var arrayName: [[…[Type]…]] = arrayValue
```

Масиви

OBJ-C

Objective-C використовує літерал масиву наступного вигляду:

```
@[
    value1,
    value2,
    value3,
    ...,
    valueN
  ];
```

SIGMA SOFTWARE UNIVERSITY

Масиви

Swift використовує літерал масиву наступного вигляду:

[value1, value2, value3, …, valueN_1]

SIGMA SOFTWARE UNIVERSITY

Масиви: Декларація

```swift
//: Arrays. Multidimension

import Foundation

typealias Element = Int

// Declaring single dimension array

let array1A: Array<Element> // 1-dimension array
let array1B: [Element] // 1-dimension array, short (preferable) syntax

// Declaring multi dimension array

let matrix2A: Array<Array<Element>> // 2-dimension array
let matrix2B: [[Element]] // 2-dimension array, short (preferable) syntax

let matrix3A: Array<Array<Array<Element>>> // 3-dimension array
let matrix3B: [[[Element]]] // 3-dimension array, short (preferable) syntax
```

SIGMA SOFTWARE UNIVERSITY

Масиви: Ініціалізація

```swift
//: Arrays. Initialization

import Foundation

typealias Element = Int

// Declaring single dimension array

let array1A: Array<Element> = Array<Element>(arrayLiteral: 1, 2, 3)                    [1, 2, 3]
let array1B: Array<Element> = [1, 2, 3]                                                [1, 2, 3]
let array1C: [Int] = [1, 2, 3]                                                         [1, 2, 3]
let array1D = [1, 2, 3] // preferable                                                  [1, 2, 3]

// Declaring multi dimension array

let matrix2A: Array<Array<Element>> = Array<Array<Element>>(arrayLiteral: Array<       [[1, 2, 3], [3, 2, 1]]
    Element>(arrayLiteral: 1, 2, 3), Array(arrayLiteral: 3, 2, 1))
let matrix2B: [[Element]] = [ [1, 2, 3], [3, 2, 1] ]                                   [[1, 2, 3], [3, 2, 1]]
let martix2C = [ [1, 2, 3], [3, 2, 1] ] // preferable                                  [[1, 2, 3], [3, 2, 1]]

let matrix3A: Array<Array<Array<Element>>>
let matrix3B: [[[Element]]] = [ [ [ 1, 2, 3 ], [3, 2, 1] ], [ [4, 5, 6], [7, 8,       [[[...], [...]], [[...], [...]]]
    9] ] ] // preferable
```

SIGMA SOFTWARE UNIVERSITY

Масиви: Copy-on-Write

Компілятор оптимізую процес копіювання масиву в інший масив.

Copy-On-Write - це технологія оптимізації процесу копіювання.

Насправді копіювання не відбувається до тих пір, поки не відбудеться перша зміна у новоствореному масиві-копії.

Масиви: Індексація

Swift/Objective-C індексує масив починаючи з 0 до N-1.

Операція звернення до елемента за індексом називається subscripting.

Оператор звернення до елемента за індексом називається subscript та має вигляд:

array[*index*]

Масиви: Індексація

```
//: Arrays. Subscripts

let arrayA: Array<Int> = []                                  []
let arrayB: Array<Int> = []                                  []
let arrayC: Array<Int> = []                                  []
var arrayD: Array<Array<Int>> = []                           []

let arrayE: [Int] = []                                       []
let arrayF: [Int] = []                                       []
let arrayG: [Int] = []                                       []
var arrayH: [[Int]] = []                                     []

let aNumber: Int = 5                                         5
let anArray: [Int] = [5]                                     [5]

arrayD.append(anArray)                                       [[5]]
arrayH.append(anArray)                                       [[5]]

arrayD.count                                                 1
arrayH.count                                                 1

arrayD[0] // anArray                                         [5]
arrayH[0] // anArray                                         [5]

arrayD[0][0] // aNumber                                      5
arrayH[0][0] // aNumber                                      5
```

SIGMA SOFTWARE UNIVERSITY

Масиви: Optionals

```
//: Arrays. Optionals

let arrayA: Array<Int> = []                                  []
let arrayB: Array<Int>? = []                                 []
let arrayC: Array<Int?>? = []                                []
var arrayD: Array<Array<Int?>?>? = []                        []

let arrayE: [Int] = []                                       []
let arrayF: [Int]? = []                                      []
let arrayG: [Int?]? = []                                     []
var arrayH: [[Int?]?]? = []                                  []

let aNumber: Int? = 5                                        5
let anArray: [Int?]? = [5]                                   [{Some 5}]

arrayD?.append(anArray)                                      ()
arrayH?.append(anArray)                                      ()

arrayD?.count                                                1
arrayH?.count                                                1

arrayD?[0] // anArray                                        [{Some 5}]
arrayH?[0] // anArray                                        [{Some 5}]

arrayD?[0]?[0] // aNumber                                    5
arrayH?[0]?[0] // aNumber                                    5
```

SIGMA SOFTWARE UNIVERSITY

Масиви: Розмір та ємність

```
1  //: Arrays. Growth and Capacity
2
3  var capacity: Int
4  var count: Int
5  var array = [Int]()
6
7  capacity = array.capacity
8  count = array.count
9
10 array += [ 1 ]
11
12 capacity = array.capacity
13 count = array.count
14
15 array += [ 2, 3 ]
16
17 capacity = array.capacity
18 count = array.count
19
20 array += [ 4, 5, 6 ]
21
22 capacity = array.capacity
23 count = array.count
24
25 array = []
26 array.reserveCapacity(100)
27
28 capacity = array.capacity
29 count = array.count
```

```
[]

0
0

[1]

2
1

[1, 2, 3]

4
3

[1, 2, 3, 4, 5, 6]

8
6

[]
[]

100
0
```

SIGMA SOFTWARE UNIVERSITY

Масиви: Операція над масивами

```
//: Arrays

var array = [1, 2, 3, 4, 5, 6, 7, 8]                                    [1, 2, 3, 4, 5, 6, 7, 8]

let capacity = array.capacity                                           8
let count = array.count                                                 8
let hasThree = array.contains(3)                                        true
let slice = array.dropFirst()                                           [2, 3, 4, 5, 6, 7, 8]
let equalToArray = array.elementsEqual([1, 2, 3])                       false
let equalToRange = array.elementsEqual(1...5)                           false
let startIndex = array.startIndex                                       0
let endIndex = array.endIndex                                           8
let first = array.first // Optional                                     1
let last = array.last // Optional                                       8
let indicies = array.indices // 0...4                                   0..<8
let isEmpty = array.isEmpty                                             false
let max = array.maxElement()                                            8
let min = array.minElement()                                            1
let startsWithArray = array.startsWith([1, 2, 3])                       true
let splitedArray = array.split(4) // Array<ArraySlice<Int>>             [[1, 2, 3], [5, 6, 7, 8]]
let reveresedArray: [Int] = array.reverse()                             [8, 7, 6, 5, 4, 3, 2, 1]

// Mutating

let removedElement = array.removeAtIndex(endIndex.advancedBy(-1))       8
let firstRemovedElement = array.removeFirst()                           1
let lastRemovedElement = array.removeLast()                             7

array.insert(0, atIndex: startIndex)                                    [0, 2, 3, 4, 5, 6]
array.removeFirst(2)                                                    [3, 4, 5, 6]
array.removeRange(0...1)                                                [5, 6]
array.replaceRange(0...1, with: [1, 2])                                 [1, 2]
array.sort()                                                            [1, 2]
```

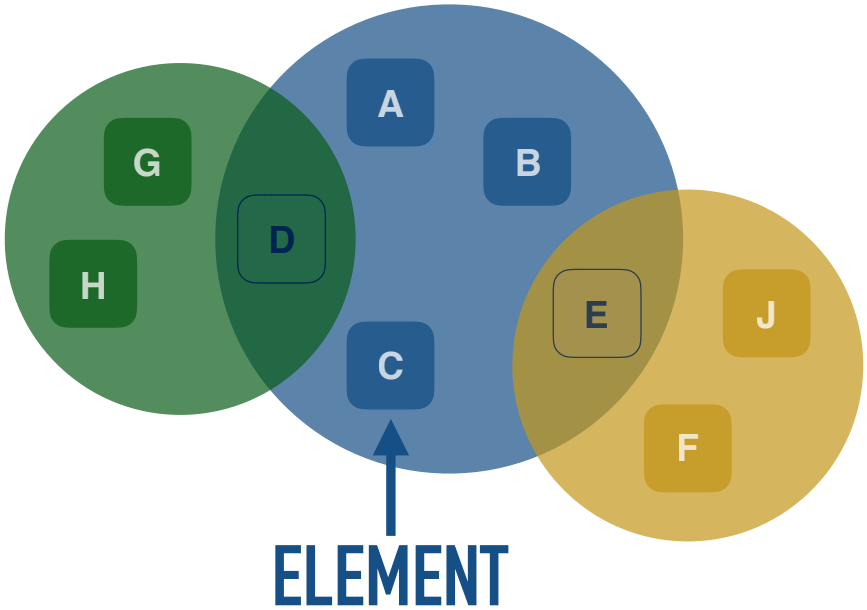
SIGMA SOFTWARE UNIVERSITY

Колекції

Колекції
Масиви
Множини
Словники
Інші колекції

Множини

Множина - *не*впорядкований набір фіксованої кількості однотипних *унікальних* елементів, що є предсталенням математичної множини.

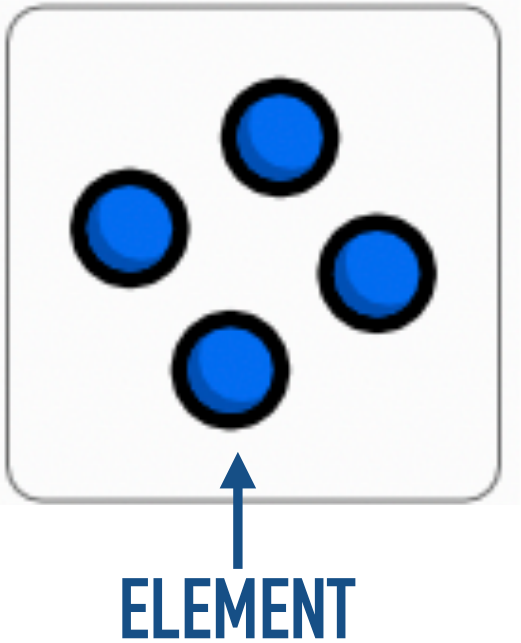


SIGMA SOFTWARE UNIVERSITY

Множини

OBJ-C

Objective-C надає NSSet, NSMutableSet, NSOrderedSet, NSCountedSet



SIGMA SOFTWARE UNIVERSITY

Множини

Swift визначає множину, як тип за значенням.

Множини

Swift надає підримку вбудованого класу Set, що представляє множину.
Формат опису множини на Swift має наступний вигляд:

let aSetName: Set<Set<…Set<Type>…>> = aSetValue
var aSetName: Set<Set<…Set<Type>…>> = aSetValue

Множини: Hashable

Swift/Objective-C вимагає, щоб елементи множини визначали т. зв. хеш-значення (hash value).

Такі елементи мають представляти тип, що реалізовує протокол Hashable.

Множини: Hashable

```
//: Sets. Hashable

let anInteger = 1                                                    1
let theIntegerHashValue = anInteger.hashValue                        1

let aDouble = 3.14                                                   3.14
let theDoubleHashValue = aDouble.hashValue                           4614253070214989087

let aCharacter: Character = "a"                                      "a"
let theChararcterHashValue = aCharacter.hashValue                    4799450059485595655

let aString = "a"                                                    "a"
let theStringHashValue = aString.hashValue                           4799450059485595655

let aTuple = (anInteger, aDouble, aCharacter, aString)               (.0 1, .1 3.14, .2 "a", .3 "a")
let theTupleHashValue = aTuple.0.hashValue ^ aTuple.1.hashValue ^    4614253070214989086
    aTuple.2.hashValue ^ aTuple.3.hashValue

let aSet = Set<Int>(arrayLiteral: anInteger)                         {1}
let bSet = Set<Double>(arrayLiteral: aDouble)                        {3.14}
let cSet = Set<Character>(arrayLiteral: aCharacter)                  {"a"}
let dSet = Set<String>(arrayLiteral: aString)                        {"a"}
```

SIGMA SOFTWARE UNIVERSITY

Множини

```
1  //: Sets
2
3  let aSet: Set<Int> // a (regular) set
4  let bSet: Set<Set<Int>>
5  var cSet: Set<String>
6
7  aSet = Set<Int>(arrayLiteral: 1, 2, 3) // not ordered                    {2, 3, 1}
8  bSet = [ [1, 2, 3], [1, 2, 3, 4] ]                                      {{2, 3, 1}, {2, 3, 1, 4}}
9  cSet = Set<String>(["a", "b", "c"]) // not ordered                       {"b", "a", "c"}
10
11 let hasTwo = aSet.contains(2)                                           true
12 let count = aSet.count                                                  3
13 let equalsToArray = aSet.elementsEqual([2, 3, 1])                       true
14 let startIndex = aSet.startIndex // an index                            SetIndex<Int>
15 let endIndex = aSet.endIndex // an index                                SetIndex<Int>
16 let first = aSet.first                                                  2
17 let min = aSet.minElement()                                             1
18 let max = aSet.maxElement()                                             3
19 let indexOfTwo = aSet.indexOf(2)                                        SetIndex<Int>
20 let indicies = Array(aSet.indices) // array of SetIndex<Int>            [{{...}}, {{...}}, {{...}}]
21 let isEmpty = aSet.isEmpty                                              false
22 let startWithTwo = aSet.startsWith([2])                                 true
23 let element = aSet[startIndex] // subscript                             2
24
25 // Mutating
26
27 cSet.insert("d")                                                        {"b", "a", "d", "c"}
28 cSet.insert("a")                                                        {"b", "a", "d", "c"}
29
30 var removedElement = cSet.remove("f")                                   nil
31
32 removedElement = cSet.remove("d")                                       "d"
33 removedElement = cSet.removeAtIndex(cSet.startIndex)                    "b"
```

SIGMA SOFTWARE UNIVERSITY

Множини

```
1  //: Sets. Operations
2
3  let aSet = Set<String>([ "a", "b", "c", "d" ])                    {"b", "a", "d", "c"}
4  let bSet = Set<String>([ "c", "d", "e" ])                         {"d", "c", "e"}
5
6  let intersection = aSet.intersect(bSet) // common elements        {"d", "c"}
7  let aSubstruction = aSet.subtract(bSet) // unique elements in aSet {"b", "a"}
8  let bSubstruction = bSet.subtract(aSet) // unique elements in bSet {"e"}
9  let exlusion = aSet.exclusiveOr(bSet) // unique elements in both  {"b", "e", "a"}
10 let union = aSet.union(bSet) // all elements                      {"b", "e", "a", "d", "c"}
11
12 var aMutableSet = aSet                                            {"b", "a", "d", "c"}
13
14 // Mutating
15
16 aMutableSet.intersect(bSet) // common elements                    {"d", "c"}
17 aMutableSet.subtract(bSet) // unique elements in aSet             {"b", "a"}
18 aMutableSet.subtract(aSet) // unique elements in bSet             []
19 aMutableSet.exclusiveOr(bSet) // unique elements in both          {"b", "e", "a"}
20 aMutableSet.union(bSet) // all elements                          {"b", "e", "a", "d", "c"}
```

Множини

```
1  //: Sets. Operations
2
3  let aSet = Set<Int>(arrayLiteral: 1, 2, 3, 4, 5)                    {5, 2, 3, 1, 4}
4  let bSet = Set<Int>(arrayLiteral: 3, 4, 5)                          {5, 3, 4}
5  let cSet = Set<Int>(arrayLiteral: 6, 7)                             {6, 7}
6
7  // subset
8  let aSubsetOfB = aSet.isSubsetOf(bSet)                              false
9  let bSubsetOfA = bSet.isSubsetOf(aSet)                              true
10
11 // subset, but not equal
12 let aStrictSubsetOfB = aSet.isStrictSubsetOf(bSet)                   false
13 let bStrictSubsetOfA = bSet.isStrictSubsetOf(aSet)                  true
14 let aStrictSubsetOfA = aSet.isStrictSubsetOf(aSet)                  false
15 let aStrictSubsetOfC = cSet.isStrictSubsetOf(aSet)                  false
16
17 // superset
18
19 let aSupersetOfB = aSet.isSupersetOf(bSet)                          true
20 let bSupersetOfA = bSet.isSupersetOf(aSet)                          false
21
22 // superset, but not equal
23
24 let aStrictSupersetOfB = aSet.isStrictSupersetOf(bSet)              true
25 let bStrictSupersetOfA = bSet.isStrictSupersetOf(aSet)              false
26 let aStrictSupersetOfA = aSet.isStrictSupersetOf(aSet)              false
27 let aStrictSupersetOfC = aSet.isStrictSupersetOf(cSet)              false
28
29 // disjoint (neither subset, nor supperset: no common elements)
30
31 let aIsDisjointOfB = aSet.isDisjointWith(bSet)                      false
32 let bIsDisjointOfA = bSet.isDisjointWith(aSet)                      false
33 let aIsDisjointOfC = aSet.isDisjointWith(cSet)                      true
```

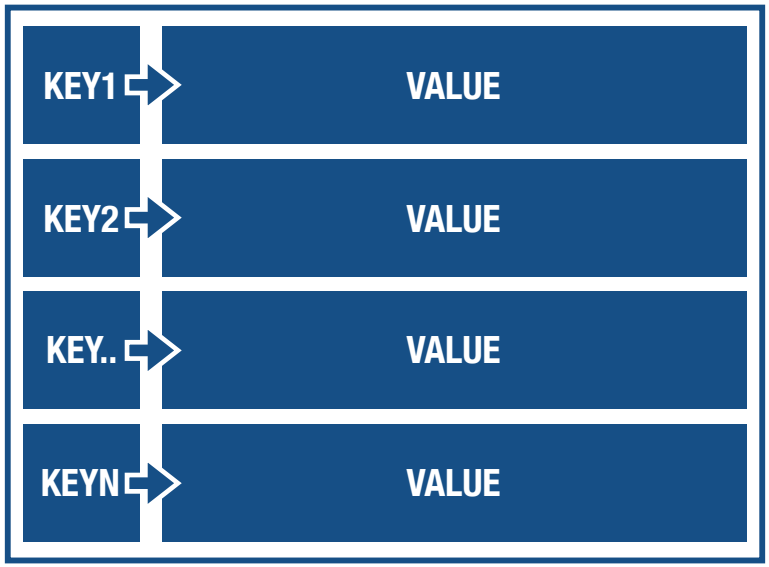
Колекції

Колекції
Масиви
Множини
Словники
Інші колекції

SIGMA SOFTWARE UNIVERSITY

Словники

Словник - *не*впорядкований набір фіксованої кількості однотипних елементів, що представлені парами типу “ключ-значення” (key-value), де ключ визначає т. зв. хеш-значення (hash value).



Словники

Swift визначає словник, як тип за значенням.

Словники

OBJ-C

Objective-C надає підримку вбудованого класу NSictionary, NSMutableDictionary.
Формат опису словника на Objective-C має наступний вигляд:

```
NSDictionary<Type:Type> *dictionaryName = @{key : dictionaryValue};

NSMutableDictionary<Type:Type> *dictionaryName = [@{key : dictionaryValue}
mutableCopy];
NSMutableDictionary<Type:Type> *finalDictionary = [NSMutableDictionary
dictionaryWithObjectsAndKeys: Object,@"value", nil];
```

Словники

Swift надає підримку вбудованого класу Dictionary, що представляє словник. Формат опису словника на Swift має наступний вигляд:

```
let dictionaryName: Dictionary<Key, Dictionary<Key, …>> = dictionaryValue
var dictionaryName: Dictionary<Key, Dictionary<Key, …>> = dictionaryValue
```

Словники

Swift надає підримку вбудованого класу Dictionary, що представляє словник. Формат опису словника на Swift має наступний вигляд:

let dictionaryName: [Key : [Key : […]]] = dictionaryValue
var dictionaryName: [Key : [Key : […]]] = dictionaryValue

Словники

OBJ-C

Objective-C використовує літерал словника наступного вигляду:

```
@{
    key0 : value0,
    key1 : value1,
    …,
    keyN : valueN
};
```

SIGMA SOFTWARE UNIVERSITY

Словники

Swift використовує літерал словника наступного вигляду:

[key0 : value, key1 : value, …, keyN_1 : value]

SIGMA SOFTWARE UNIVERSITY

Словники

```
1  //: Dictionaries                                                                                    
2                                                                                                    
3  let aDictionary: Dictionary<Int, String>                                                          
4  let bDictionary: [ Int : String ]                                                                 
5  let cDictionary = Dictionary<Int, String>(dictionaryLiteral: (1, "1"), (2, "2"))                  [2: "2", 1: "1"]
6  let dDictionary = [ 4 : "Four", 5 : "Five" ]                                                      [5: "Five", 4: "Four"]
7  var mDictionary = dDictionary                                                                     [5: "Five", 4: "Four"]
8                                                                                                    
9  let count = cDictionary.count                                                                     2
10 let startIndex = cDictionary.startIndex                                                           DictionaryIndex<Int, String>
11 let endIndex = cDictionary.endIndex                                                               DictionaryIndex<Int, String>
12 let first = cDictionary.first // an optional tuple                                                (.0 2, .1 "2")
13 let indexOfFour = dDictionary.indexForKey(4) // an optional DictionaryIndex                       DictionaryIndex<Int, String>
14 let isEmpty = dDictionary.isEmpty                                                                 false
15 let indicies = Array(dDictionary.indices) // an array of DictionaryIndex                          [{{...}}, {{...}}]
16 let keys = Array(dDictionary.keys) // an array of keys                                            [5, 4]
17 let values = Array(dDictionary.values) // an array of values                                      ["Five", "Four"]
18                                                                                                   
19 if let index = indexOfFour {                                                                      
20     let removedValue = mDictionary.removeAtIndex(index) // a (non-optional) tuple                 (.0 4, .1 "Four")
21 }                                                                                                 
22                                                                                                   
23 let removedValue = mDictionary.removeValueForKey(4)                                              nil
24 mDictionary[6] = "Ten" // a subscript as "append"                                                 "Ten"
25 mDictionary[6] = "Seven" // a subscript as "update", if exists                                   "Seven"
26 mDictionary.updateValue("Six", forKey: 6)                                                        "Seven"
27                                                                                                   
28 let reversedDictionary = mDictionary.reverse()                                                   [(.0 6, .1 "Six"), (.0 5, .1 "Five")]
29                                                                                                   
30 let multiDictionary: Dictionary<Int, Dictionary<String, Double>> = [ Int : [ String : Double ] ]()  [:]
31 let emptyDictionary: [Int : [ Int : Int ]] = [:]                                                 [:]
```

SIGMA SOFTWARE UNIVERSITY

Колекції

Колекції
Масиви
Множини
Словники
Інші колекції

SIGMA SOFTWARE UNIVERSITY

Інші колекції

OBJ-C

NSOrderedSet
NSCountedSet
NSMapTable
NSHashTable
NSPointerArray

Інші колекції

ArraySlice
CollectionOfOne
ContinousArray
EmptyCollection
Range

Інші колекції: ArraySlice

```
//: ArraySlice

let array = [ 1, 2, 3, 4, 5 ]                                   // [1, 2, 3, 4, 5]
let range = 1...5                                               // 1..<6

let arraySlice = array.dropFirst(3)                             // [4, 5]
let rangeSlice = range.dropFirst(3)                             // Slice<Range<Int>>

print(arraySlice, terminator: "")                               // "[4, 5]"
print(rangeSlice, terminator: "")                               // "Slice<Range<Int>>(startIndex: 4, endIndex: 6, _base: Range(1..<6))"

let equalToArray = arraySlice == [4, 5]                         // true
let equalToRange = rangeSlice.elementsEqual(4...5) // no == defined // true

var mutableArray = array                                        // [1, 2, 3, 4, 5]
var mutableArraySlice = mutableArray.dropFirst(3)               // [4, 5]

print("Indices: \(mutableArraySlice.indices)")                  // "Indices: 3..<5\n"

let indexOf3 = mutableArraySlice.indexOf(4) // element at index == 3 // 3

mutableArray.removeAtIndex(4) // element at index == 5          // 5

//let mutableArraySlice = mutableArray.dropFirst(3)

print(mutableArray)                                             // "[1, 2, 3, 4]\n"
print(mutableArraySlice)                                        // "[4, 5]\n"

mutableArraySlice.removeAtIndex(3)                              // 4

print("Indices: \(mutableArraySlice.indices)")                  // "Indices: 3..<4\n"

print(mutableArray)                                             // "[1, 2, 3, 4]\n"
print(mutableArraySlice)                                        // "[5]\n"
```

SIGMA SOFTWARE UNIVERSITY

Інші колекції: Range

```
//: Ranges

// empty range
var range = Range<Int>(start: 0, end: 0)                    0..<0

let endIndex = range.endIndex                               0
let startIndex = range.startIndex                           0

var isEmptyRange = startIndex == endIndex                   true

// non-empty range
let singleRangeDeclaration = 0..<10                         0..<10
range = Range(start: 0, end: 10)                            0..<10

isEmptyRange = startIndex == endIndex                       true
isEmptyRange = range.isEmpty                                false

// range operations

let hasFive = range ~= 5                                    true
let hasFiveAgain = range.contains(5)                        true
let count = range.count                                     10

// enumerating via loop (will be discussed later)

let slice = range.dropFirst()                               Slice<Range<Int>>
print("\(slice)")                                           "Slice<Range<Int>>(startIndex: 1, endIndex: 10, _base: Range(0..<10))\n"

var equalToSequence = range.elementsEqual([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])   true
equalToSequence = range.elementsEqual(0..<10)               true

let indexOf5 = range.indexOf(5) ?? -1                       5
```

SIGMA SOFTWARE UNIVERSITY

Інші колекції: Range

```
13 range = Range(start: 0, end: 10)                                  0..<10
14
15 isEmptyRange = startIndex == endIndex                             true
16 isEmptyRange = range.isEmpty                                      false
17
18 // range operations
19
20 let hasFive = range ~= 5                                          true
21 let hasFiveAgain = range.contains(5)                              true
22 let count = range.count                                          10
23
24 // enumerating via loop (will be discussed later)
25
26 let slice = range.dropFirst()                                     Slice<Range<Int>>
27 print("\(slice)")                                                 "Slice<Range<Int>>(startIndex: 1, endIndex: 10, _base: Range(0..<10))\n"
28
29 var equalToSequence = range.elementsEqual([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])  true
30 equalToSequence = range.elementsEqual(0..<10)                     true
31
32 let indexOf5 = range.indexOf(5) ?? -1                            5
33
34 let rangeIndicies = range.indices                                 0..<10
35
36 if let first = range.first, last = range.last {
37     print(".first = \(first), .last = \(last)", terminator: "")   ".first = 0, .last = 9"
38 }
39
40 if let min = range.minElement(), max = range.maxElement() {
41     print(".min = \(min), .max = \(max)", terminator: "")         ".min = 0, .max = 9"
42 }
43
44 // collection of indicies
45 // (-99...100)[0] == 0
```

SIGMA SOFTWARE UNIVERSITY

Інші колекції: Додаткові

```
//: Other Collections

let collectionOfOne = CollectionOfOne<Int>(1)                    CollectionOfOne<Int>
let hasOne = collectionOfOne.contains(1)                         true
let count = collectionOfOne.count                                1

if let index = collectionOfOne.indexOf(1) {
    let element = collectionOfOne[index]                         1
}

if let index = collectionOfOne.indexOf(2) {
    let element = collectionOfOne[index]
} else {
    print("This is collection of one only.")                     "This is collection of one only.\n"
}

let emptyCollection = EmptyCollection<String>()                  EmptyCollection<String>
let emptyCollectionCount = emptyCollection.count                 0

if let first = emptyCollection.first {
    print(first)
} else {
    print("This is empty collection.")                           "This is empty collection.\n"
}

let array = Array<String>(emptyCollection)                       []
let anotherArray = Array(collectionOfOne)                        [1]

anotherArray.dynamicType                                         Array<Int>.Type
```

SIGMA SOFTWARE UNIVERSITY


SIGMA
Software
UNIVERSITY